



PROCEEDINGS

Workshop on Large Installation Systems Administration

Monterey, CA
November 17-18, 1988

The USENIX Association

The USENIX Association is a non-profit association of individuals and institutions interested in fostering innovation and sharing ideas, software, and experience where UNIX and UNIX-like systems and the C programming language are concerned. USENIX sponsors technical conferences and workshops and an annual vendor exhibition; publishes *login*: (a bi-monthly newsletter) and *Computing Systems* (a technical quarterly); distributes 2.10BSD and the 4.3BSD manuals; and serves as coordinator of a software exchange for appropriately licensed members.

The individual and institutional members of USENIX are interested in problem-solving with a practical bias, with research that works, and with timely responsiveness within a community made up of executives and managers, programmers, and academics.

Future Events

USENIX 1989 Winter Conference
San Diego, Jan. 30-Feb. 3, 1989

EUUG Spring Conference
Brussels, Apr. 10-14, 1989

Long-term USENIX & EUUG Schedule

Jun 12-16 '89 Hyatt Regency, Baltimore
Sep 18-22 '89 Vienna, Austria
Jan 22-26 '90 Omni Shoreham, Washington, DC
Apr 23-27 '90 Munich, W. Germany
Jun 11-15 '90 Marriott Hotel, Anaheim
Jan 21-25 '91 Dallas
Jun 10-14 '91 Opryland, Nashville
Jan 20-24 '92 Hilton Square, San Francisco
Jun 8-12 '92 Marriott, San Antonio

USENIX Workshops in 1989

Transaction Processing
Distributed Systems
Software Management
Graphics V

Check your local paper for further details.

For additional copies of these proceedings write

USENIX Association
P.O. Box 2299
Berkeley, CA 94710 USA

The price is \$8.
Outside the U.S.A. please add
\$7 per copy for postage (via air printed matter).

© 1988 USENIX Association. All Rights Reserved.
This volume is published as a collective work.
Rights to individual papers remain
with the author or the author's employer.

UNIX is a registered trademark of AT&T.

Program and Table of Contents

Workshop on Large Installation Systems Administration November 17-18, 1988

Wednesday, November 16

I. Layton Room
(Double Tree Inn)

Registration and Get-together

7:00 - 10:00

Thursday, November 17

Steinbeck Forum
(Monterey Conference Center)

Opening Remarks

9:00 - 9:45

Alix Vasilatos, Workshop Chair

Keynote Speech

The Evolving Role of the System Administrator

Rob Kolstad, Prisma, Inc.

Break

9:45 - 10:15

Accounts and Accounting

10:15 - 11:15

Password Administration for Multiple Large Scale Systems

1

Bruce H. Hunter, Intel Corporation

Administration of Network passwd files and NFS File Access

3

Deb Lilly, John Fluke Mfg. Co.

Project Accounting on a Large-Scale UNIX System

7

W. H. Gray & A. K. Powers, Idaho National Engineering Lab.

UNIX Login Administration at Bellcore

13

Wayne C. Connelly, Bell Communications Research

Tools I

11:15 - 12:00

Makealiases - a mail aliasing system

17

Gretchen Phillips, SUNY@Buffalo

Don Gworek, Sun Microsystems

A Notice Capability for UNIX

21

Michael A. Erlinger, Harvey Mudd College

A Batching System for Heterogeneous UNIX Environments	23
Helen E. Harrison, Microelectronics Center of North Carolina	
<i>Lunch</i>	12:00 - 1:30
<i>Tools II</i>	1:30 - 2:15
A Lazy Man's Guide to UNIX Systems Administration	25
Bjorn Satdeva, /sys/adm, inc.	
Netdump: A Tool for Dumping Filesystems	27
D. Ryan Hawley, Sun Federal Microsystems, Inc.	
Combining Two Printing Systems Under a Common User Interface	29
Dave Goldberg, MITRE Corporation	
<i>Break</i>	2:15 - 2:45
<i>Backup and Restore</i>	2:45 - 4:00
A Flexible Backup System for Large Disk Farms, or What to do with 20 Gigabytes	33
Helen E. Harrison, Microelectronics Center of North Carolina	
The Andrew Backup System	35
Steve Hecht, Carnegie-Mellon University	
BUMP - The BRL/USNA Migration Project	39
Mike Muuss, BRL, Terry Slattery, USNA, & Don Merritt, BRL	
A Simple Incremental File Backup System	41
Patricia E. Parseghian, Princeton University	
Backup at Ohio State	43
Elizabeth Zwicky, Ohio State University	
<i>Panel — Backup: Variations on a Theme</i>	4:00 - 5:00
Mike Muuss, BRL, Chair	
Andrew Hume, AT&T Bell Laboratories	
Rob Kolstad, Prisma, Inc.	
Ed Gould, mt Xinu	
Pat Parseghian, Princeton University	
<i>No-host Reception — I. Layton Room</i>	6:00 - 8:00
<i>Birds of a Feather Sessions — Steinbeck Forum</i>	8:15 - 11:00
Root Journal	
InfoPro Systems	

Friday, November 18	Steinbeck Forum
<i>Planning and Standardization</i>	9:00 - 10:15
Transitioning Users to a Supported Environment	45
Earl W. Norwood, III, Hewlett-Packard Laboratories	
Making a Large Network Reliable	47
Steve Simmons, Inland Sea Software, Ltd.	
Update on Systems Administration Standards	49
Steve Carter, Bell Communications Research	
Standards and Guidelines for UNIX Workstation Installations	51
James Hayes, Advanced Micro Devices	
Computer Aided Capacity Planning of a Very Large Information Management System	63
W. Bruce Watson, Lawrence Livermore National Laboratory	
<i>Break</i>	10:15 - 10:45
<i>Works in Progress</i>	10:45 - 12:00
<i>Lunch</i>	12:00 - 1:30
<i>Distributed Systems</i>	1:30 - 3:00
System Administration in the Andrew File System	67
Marybeth Schultz Cyganik, Carnegie-Mellon University	
Service Management at Project Athena	71
Daniel E. Geer, Jr., MIT	
Concurrent Access Licensing and NLS	73
David Ortmeyer, Apollo Computer	
A Subscription-Oriented Software Package Update Distribution System (SPUDS)	75
Ola Ladipo, AT&T Bell Laboratories	
System Administration and Maintenance of Fully Configured Workstations	79
Robert E. Van Cleef, General Electric Corporation	
Capacity Testing a HYPERchannel-Based Local Area Network	83
W. Bruce Watson, Lawrence Livermore National Laboratory	
<i>Break</i>	3:00 - 3:30
<i>Panel — World Inter-Site Network Connectivity</i>	3:30 - 4:30
John Quarterman, TIC, Chair	
Rick Adams, UUNET	
Ron Hoffmann, MIT	
<i>Open Discussion: Systems Administration in the 1990s</i>	4:30 - 5:00

Workshop Chair:

Alix Vasilatos
MIT Project Athena
E40-357
1 Amherst Street
Cambridge, MA 02139
(617) 253-0121
alix@athena.MIT.EDU

USENIX Workshop Coordinator:

Judith H. DesHarnais

USENIX Workshop Liaison:
Rob Kolstad

Proceedings Production:
Tom Strong

Password Administration for Multiple Large Scale Systems

Bruce H. Hunter

Intel Corporation

My installation utilizes several hundred desktop engineering systems running SunOS and other UNIX variations networked to mini computers running ULTRIX. All of this is networked to IBM 3090 600E mainframes running UTS UNIX. User sites are located through the American Southwest and overseas. The operating system network strategy is UNIX and Ethernet TCP/IP.

Maintaining password file security, uniformity, and portability on a complex international system is not without its problems. Unless the password files of all systems have matching lognames with user ID numbers for each user, files transferred from one system to another quickly lose their ownership. With literally hundreds of users moving from project to projects, the password file quickly becomes outdated. It can fill with expired user IDs unless diligently monitored. Password aging adds another problem by leaving /etc/passwd open to invasion by devious users. In time the password file fills with expired logins which represent a threat to security. They also slow down the time required to log in.

There are even more problems unique to this installation. Frequent file system reorganizations to accommodate the creation of new project groups and the deletion of old ones leave logins without a home directory. Running a batch system requires a batch-UNIX system to be attached to each interactive-UNIX system. The batch system password files must be identical to the interactive system files except they must not allow an interactive login.

A few of the solutions to these problems are:

1. Maintaining a central password data base reachable by all sites and systems
2. A program to add new users that uses the central data base and updates it
3. Procedures, manual pages and software to search the password files and report all expired and blocked passwd entries as well as would-be logins with no home directory
4. Using existing software like standard /etc/pwck
5. Software, manual pages and procedures to create batch password files with blocked password fields
6. Procedures for modifying /etc/passwd without leaving this critical file open for writing and thus endangering the system

The final step in these solutions is periodically checking and correcting all password files on all systems. A side benefit is the updating of group and identity* files.

* UTS systems maintain a file called /etc/identity that gives useful user information such as site location, phone number, department number and other useful information.

Administration of network passwd files and NFS file access

Deb Lilly

John Fluke Mfg. Co.
deb@tc.fluke.COM

PROBLEM

When a filesystem is exported, NFS file access is not subject to password authorization (as required with remote logins and remote shells). The user and group id's on the local system (which NFS-mounted the filesystem) are honored to determine permissions for accessing the remote (exported) files. Therefore, it is extremely important to ensure that user and group id's are used consistently on all systems which share NFS files.

How can we administer accounts and NFS file access on a network, especially when the machines are administered by different departments?

Abbreviations used in this paper:

gid	group id number
NFS	Network File System
NIR	local Network Information Registry
O/S	operating system (UNIX on most of our machines)
TC	local Technical Computing dept (uses YP domain 'tc')
uid	user id number
YP	Yellow Pages network service

Characteristics of our Site

Our network has 120 hosts on one extended Ethernet, including VAX, SUN, IBM equipment, and 16 PC's running pc-nfs. Most of the systems are SUNs running UNIX (about 65 diskless clients and 10 fileservers). A central Technical Computing department (TC) administers the network and most of the equipment. However, several engineering departments administer their own equipment, yet need to interface with the TC network and NFS-mount TC filesystems. TC is self-supporting; it charges back its costs to all users.

We use the Yellow Pages (YP) and have added several local YP maps. Additional databases exist for tracking chargeback information, maintenance contracts, which clients are on a fileserver, etc.

SOLUTION

We have: (1) established a centralized Network Information Registry, (2) established the policies listed below, and (3) designed a relational database to integrate the various administrative databases (including several Yellow Pages maps) and to reduce duplication of information.

Network Registry Information Service

TC provides a local Network Information Registry (NIR) service so that system administrators at our site can request and register loginnames, uids, groups, gids, hostnames, and internet addresses. This ensures that the name and/or number is unique in the tc YP domain and will not be used for another entity on any systems which NFS-mount TC filesystems. For example, when a new account is created, the loginname and uid are checked for uniqueness in the NIR as well as in the YP passwd map and /etc/passwd file entries.

Policies

The TC department has set the following policies in response to the problems we have encountered.

- Each user has the same loginname and uid across all UNIX systems. This is necessary to ensure that user X on one machine is the same user X who owns a file that is NFS-mounted from another machine.
- We map our uids and gids at O/S upgrade time. The VAX and SUN machines don't completely agree about system accounts, so modifications are needed to keep accounts and groups consistent on all our UNIX systems. As more vendors are involved, the problem of conflicting uids/gids increases.
- No passwd entry can have a null password. A null password means that anyone can login on the account (the system does not even prompt for a password); this is a serious security risk for any network with inbound dialup lines and with proprietary files which may have world read access.
- TC filesystems are exported (for NFS mounting) ONLY to hosts in the tc domain or in a netgroup of trusted hosts. Otherwise, we cannot ensure against unauthorized access to NFS-mounted files.
- Only machines administered by TC can be YP servers for the domain tc. We have many machines depending on YP; there is no way to control to which machine a system binds for YP. If we allowed machines administered by other departments to serve YP to our customers, we would risk providing unpredictable service.
- To NFS-mount filesystems from TC, machines must either be in the tc domain or register with the NIR. A machine can join the tc domain after it has been demonstrated that no conflicts exist between passwd, group, and hosts files, vs. the tc YP domain. This may involve finding all files with certain owners and/or groups and changing owner and/or group on the files to ensure consistency and uniqueness. Typically this process must be repeated with installation of new releases of the operating system.

- The /etc/hosts.equiv files on all TC systems restrict access to "trusted hosts", which are either administered by TC or have registered with TC's NIR. This restricts remote logins and remote shells to machines on which there are no conflicting loginnames, uids, groups, gid, etc.
- Vendors are encouraged to separate O/S and application distributions, to facilitate resolving conflicts in administrative files. Vendors are strongly discouraged from requiring special loginnames, uids, absolute pathnames, or root access for installation of their applications.

Administrative Database

We are designing a relational database from which we can generate YP maps (for example: passwd, mailaliases, hosts, and ethers) and other administrative files. In addition to the data stored in the YP maps, we need to track related information for accounts and hosts. We need a mechanism to reserve loginnames, uids, group names, gids, etc., which are not in the tc YP domain. For security reasons, we may want to exclude from the tc YP domain certain logins used in a private /etc/passwd file or in another domain. If we included every registered login/uid in the tc domain, then anyone knowing the password to such an account could login on any machine on the TC network.

We currently have multiple data files which will be combined. For example, adding a new host to the network currently requires entries in YP maps hosts and ethers, in one database for charge-back information, another for maintenance contract information, and in another database that tracks which clients are served by which fileservers, etc. We are designing a hosts database to combine the YP hosts map data with additional fields: architecture, subnetwork, charge information, etc. From this database, we will generate /etc/hosts files, the appropriate YP maps, and other administrative files.

We need to generate several passwd files: private /etc/passwd files for a variety of machines (time sharing machines, fileservers, clients, and specialized machines such as our uucp host) as well as the YP passwd map. Our plan is to create a passwd database that includes all the regular /etc/passwd fields plus several additional fields: employee id, mail alias, charge info, type of account, status, whether on TC mailing list, which passwd files contain each entry, etc. It will include the loginnames and uids reserved in the NIR but not used in the YP passwd map. From this database, we will derive each /etc/passwd file needed and the source file for the YP passwd map. We plan to provide a mechanism for users to modify their "gecos" field (phone number, etc.). We will check for any changes in gecos fields before regenerating passwd files, so that user changes can be incorporated into the database.

The database is designed to allow easy access to "all" information about one user or host (for example, login name, mail alias, workstation or terminal serial number, billing information, etc.) in one place. The administrative database design will be available to interested parties.

Project Accounting on a Large-Scale UNIX¹ System²

W. H. Gray

A. K. Powers

**Idaho National Engineering Laboratory
EG&G Idaho, Inc.
Idaho Falls, ID 83415**

Introduction

"First thing we do, let's kill all the accountants."

- With apologies to Will Shakespeare³

In August of 1987, the Idaho National Engineering Laboratory (INEL) acquired a Cray X-MP/24 supercomputer. During the course of the procurement, the decision was made to go into production with the UNICOS operating system, which is Cray Research's port of UNIX System V, Release 3. Since our previous scientific computing environment was anchored by twin Control Data Corporation Cyber 176's running the NOS operating system, this course of action entailed a certain amount of culture shock for the majority of the Scientific Systems Technical Support staff and the user community.

Our limited experience with our Digital Equipment Corporation VAX 11/750 running 4.2BSD UNIX, and pre-delivery training on UNICOS led us to the conclusion that we would not get away with the primitive accounting support previously offered to our small VAX user community. Therefore, the first software project we launched in the UNICOS environment was to invent some sort of a kludge that could be done quickly and that would, at least minimally, support our large and very project-oriented scientific user base.

The Requirements

"I've got a little list, I've got a little list."

- Gilbert & Sullivan

The Mikado

The INEL's mainframe computer accounting requirements are driven by the project orientation of the majority of the users and the direction given by the Department of Energy (DOE) to recover the cost of the system's operation by charging the end user directly. Moreover, charges should be proportional to the resources utilized

1. UNIX is a trademark of AT&T Bell Laboratories

2. Work performed under the auspices of the U.S. Department of Energy, DOE Contract No. DE-AC07-76ID01570.

3. Henry VI, Part II: "..., let's kill all the lawyers."

and the level of service at which those resources are delivered. These general considerations in turn led to the following list of specific requirements.

Project orientation: A significant amount of work consists of varied projects, both small and large, which utilize the work of different individuals who are sometimes from different organizations. The same individual may simultaneously do work for different projects (*i.e.*, different "account" or "charge" codes). It is terminally cumbersome to force such individuals to use different login-names, home directories, etc. for each project they may work on. A method was required to enable a user to maintain just one home directory, have reasonable access to all his/her files and to switch easily from one account to another.

Cost recovery: system usage must be accounted for in such a way that customer's organizations and projects are charged proportionately according to their consumption of system resources. UNICOS (Release 3.0) provides enough accounting information to allow us to collect process accounting data by user-id; however there isn't any mechanism to split one user's resource consumption among different charge codes. The same observation applies to disk space; in fact, in order to do this kind of accounting for disk space, we believe that a charge code would have to be incorporated into the inode. Therefore a method was required to allow accounting for disk space so that when a user switches from one account to another, disk space usage, as well as process accounting, accrues accordingly.

Chargeable resources: in our case, these are (1) Central Processor (CPU) time, (2) Memory,⁴ (3) I/O (bytes transferred) and (4) disk space. UNICOS provides adequate facilities for collecting this data.

Charge for services: our policy is to charge proportionately according to the level of service that the user requests (and, one hopes, actually receives). *E.g.*, jobs run at a higher priority get better service than jobs run at a lower priority. Given two jobs that consume identical system resources, the job run at the higher priority should be charged more. Here, about all one can do is charge according to the "nice" value that a given process was run at; since that value is, according to UNICOS documentation, preserved in a process' accounting record.

The Kludge

Client to accountant: "How much is two and two?"

Accountant: "How much would you like it to be?"

- Anon.

In order to make one user's resource usage billable under different charge codes, the UNIX group capability was pressed into somewhat reluctant service. Under this

4. Memory usage is actually expressed as the integral of memory as a function of time; thus, the charge for memory is weighted by how long it is tied up.

scheme, when a user is registered, s/he is given a login name and a charge code, and is placed in a (newly created) group whose group name is identical with his/her login name.

The account established by this registration process is called the user's *primary account* and the login name for that account is called the *primary login name*. For example, suppose that a user is registered with the login name of *lex*, and that this (primary) account is associated with the billing code of "1066." Then, as a result of the registration process, that user also becomes the owner and sole member of group *lex*.

If user *lex* is then required to make use of a different billing code, say "1588," s/he may establish a *secondary account* associated with that billing code. It is important to note that users themselves establish secondary accounts, using a command we provide. In this way, any user can establish up to ten secondary accounts, each associated with a different billing code. These secondary accounts are real live password file entries whose login names are derived from the primary login name with "0," "1," ..., "9" appended for each secondary account so created. Therefore, in our example, user *lex* would have a secondary account under login name *lex0* associated with billing code 1588.

As the secondary account is created, the secondary login name is entered as a member of the primary account's group; in the example, *lex0* becomes a member of the group *lex*. Similarly, if secondary accounts *lex1*, *lex2*, ..., *lex9* are ever set up, they also become members of group *lex*. Secondary accounts are not given passwords; instead a command similar to *su(1)* is provided so that users may switch sessions from one account to another. For *lex* to switch from executing under his/her primary account to executing under the secondary account *lex0*, s/he has only to type

```
setact lex0
```

Since the secondary accounts really are separate user-id's, files created under them can be readily charged to different billing codes. User inconvenience is minimized by arranging that (csh) shell variable *umask* is such that files have read and execute permission for group by default. In addition, secondary accounts' home directories are rooted in the primary account's home directory. For example, if *lex*'s home directory is */usr/lex*, then *lex0*'s home directory would be */usr/lex/lex0*. In this way, the primary account has reasonable access to all of the secondary accounts' files via the group permission mechanism.

Some Implementation Details

"O what a tangled web we weave ..."
-Sir Walter Scott

Of course, some programming had to be done to support this kludge. The items below summarize the necessary effort.

nu: (new user). This is a public domain program that originated at Stanford University. It presents a menu and query driven interface to the caller, enabling a clerk or secretary to set up new accounts. Naturally, it required extensive modifications.

/etc/group: some provision had to be made to keep group-id's used for accounting purposes separated from those being used for their intended purpose. We arbitrarily allocated the range from 200 to 50,000 for accounting purposes.

creact: (create account). This program allows a user to establish a secondary account; it was designed and written from scratch.

setact: (switch to secondary account). This program was based on *su(1)*, and behaves in much the same way, except that a password is not required.

modgrp: (modify group). This is a utility that allows owners of groups to add or delete members.

valact: (validate account). This program allows a user to determine if a charge code is valid.

Experience thus Far

"Experience keeps a dear school, but a fool will learn in no other"
-Benj. Franklin

Generally, despite much grousing, complaining, threats on our lives, aspersions on our ancestry, &c, &c, this scheme seems to be working out. We have avoided placing an onerous burden on our user community at the expense of a gross inconvenience, while partially solving the problem of project and disk accounting. However, there are some serious drawbacks:

- Considerable violence is done to the normal usage of UNIX groups. Some of our user groups are large, and they already suffer from MAXGROUP limitations.⁵
- Primary and secondary accounts have to be associated with charge codes. This is accomplished using a file. Billing runs are made relatively infrequently, and they rely on this file to associate a login name with a charge code. Therefore, if a user changes the charge code associated with a login name, that charge code will be applied across the entire billing period. This care should not really be required of the user.

5. This is not as straightforward to fix as might be supposed; definitions of MAXGROUP are scattered throughout several different header files.

nu: (new user). This is a public domain program that originated at Stanford University. It presents a menu and query driven interface to the caller, enabling a clerk or secretary to set up new accounts. Naturally, it required extensive modifications.

/etc/group: some provision had to be made to keep group-id's used for accounting purposes separated from those being used for their intended purpose. We arbitrarily allocated the range from 200 to 50,000 for accounting purposes.

creact: (create account). This program allows a user to establish a secondary account; it was designed and written from scratch.

setact: (switch to secondary account). This program was based on *su(1)*, and behaves in much the same way, except that a password is not required.

modgrp: (modify group). This is a utility that allows owners of groups to add or delete members.

valact: (validate account). This program allows a user to determine if a charge code is valid.

Experience thus Far

"Experience keeps a dear school, but a fool will learn in no other"
-Benj. Franklin

Generally, despite much grousing, complaining, threats on our lives, aspersions on our ancestry, &c, &c, this scheme seems to be working out. We have avoided placing an onerous burden on our user community at the expense of a gross inconvenience, while partially solving the problem of project and disk accounting. However, there are some serious drawbacks:

- Considerable violence is done to the normal usage of UNIX groups. Some of our user groups are large, and they already suffer from MAXGROUP limitations.⁵
- Primary and secondary accounts have to be associated with charge codes. This is accomplished using a file. Billing runs are made relatively infrequently, and they rely on this file to associate a login name with a charge code. Therefore, if a user changes the charge code associated with a login name, that charge code will be applied across the entire billing period. This care should not really be required of the user.

5. This is not as straightforward to fix as might be supposed; definitions of MAXGROUP are scattered throughout several different header files.

- User's file management is complicated, because secondary accounts really are separate login names; management of the files associated therewith clashes with normal group usage.

Future Considerations

"The future lies ahead."

-Alfred E. Neumann, Casey Stengel, or maybe Yogi Berra

Is there a lesson to be learned from the necessity to invent such a monstrosity? We believe that a large-scale system, with a numerous user community, needs more capable accounting than is delivered in System V, Release 3. The following requirements should be considered for future implementations:

- Project accounting: an accounting identifier should be associated with processes, files and connect time records. A user should be able to switch the execution of his/her session from one account id to another. Files should acquire the accounting identifier attribute of the session under which they were created; however, the owner [super user only?] should have the ability to change that attribute.
- Trackability: on big, fast systems, users run big, complex jobs⁶ that burn up huge quantities of CPU time while using millions of words of memory. Such jobs cost a lot, sometimes as much as buying your very own PC or workstation, and the user wants an itemized bill. Thus, some mechanism is needed to identify the components of a job in the process accounting data.
- Service-based charging: some users' needs for quick turnaround are more urgent than others', and they are willing to pay for enhanced service. The scheduler should be flexible enough to cater to this requirement, and it should be possible to determine the level of service accorded from process accounting data.

6. Think of a job as a script run in the background.

Unix^{*} Login Administration at Bellcore

Wayne C. Connelly

**Member of Technical Staff
Bell Communications Research**

The Unix computer center at Bell Communications Research (Bellcore) maintains 43 mini-computers running Unix System V with approximately 11,000 logins. The systems are DEC VAX[@] 780s, 785s, 8650s and AT&T 3B20s. Each system is a node in a NSC Hyperchannel[#] based local area network, which in conjunction with the Bellcore developed "ins" software, allows remote command execution and file transfer. These remote capabilities are executed with the permissions of the requester of the service. It is therefore possible to execute a command remotely as "root" or other privileged id. As the center increased in size to as many as 50 systems, login administration consumed significant resources and associated problems became more frequent.

To resolve these issues the author designed and developed the "pwadm" software subsystem which performs login administration. For the purpose of this paper, login administration is defined as administering modifications to the password file and other changes necessary to ensure a user's environment corresponds to that shown in the password file. "Pwadm" was designed to meet the following requirements:

- It had to be user friendly to its various user communities to avoid having users finding it more convenient to bypass the system.
- It had to perform complete validation of input and avoid corruption of the password file and user data. Unavoidable failures had to be handled gracefully, thus instilling a high degree of confidence in its users.
- It had to be complete enough to eliminate the need to manually edit the password file.
- It needed to be convenient to complete a modification to one, several, or all password files.

The first three items are designed to ensure "pwadm" is the method of preference among those needing to perform login administration. The last has more to do with efficiency. Previous to "pwadm", adding a new login to all systems involved

^{*} Unix is a registered trademark of AT&T Bell Laboratories

[@] DEC and VAX are trademarks of Digital Equipment Corporation

[#] Hyperchannel is a trademark of Network Systems Corporation

several days of work by a clerk. Using "pwadm", such an operation is completed within a matter of several minutes.

Most often, changes to, additions, and deletions of logins are performed by clerks who, upon receipt of the proper paperwork, complete the request. System administrators also find it necessary to change the password file. "Pwadm" therefore has two interfaces available so that both communities of users have an interface available that best meets their level of expertise.

Clerks use a menu driven interface that resides on a "hub" system. Each hub has a database of free user ids(uids) and a copy of each system's password file. The database of free uids is used to select a unique uid for new logins. The copies of the password files are used to ensure the validity of the input (e.g. no duplicate login names). In a typical session, a clerk inputs data including the system upon which the change is to take effect. Before any action is taken, the input is extensively validated. If all tests are passed, a request is made to make the change on the chosen system via the remote execution capability of "ins".

Once the modification is completed, "pwadm" returns a copy of the new password file to the "hub" systems. The user of "pwadm" receives electronic mail informing them of the status of the modification. Normally confirmation is received within a few minutes depending upon network traffic. The owner of the login affected receives either paper or electronic mail, as appropriate, informing them of the change.

The menu interface is merely a front end to that normally used by the administrators. This interface is simply a set of three commands "adduser", "chguser", and "deluser" which add, change and delete logins respectively. Each command has a set of options each of which represent a field of the password file entry needed to be specified. Each command can be invoked locally to affect changes to the local password file or remotely via the "ins" network. All three commands perform extensive error checking. Some of these checks are redundant of those performed by the menu interface (when used), others can only be verified on the system on which the change is being made. For example, the validity of a login directory path can only be established on the system itself since other systems have no knowledge of another's directory structure. The example below shows one such operation:

If "sysadmin1" (who has a login on all our systems) were to leave our organization we would invoke the following command to remove the login from the systems.

```
ins +all -n -x /pkg/pwadm/bin/deluser -l sysadmin1
```

The "ins" software would ensure the deluser command is executed (-x option) on "all" systems and that mail from each would be returned containing the output of the deluser command (-n option). Upon successfully deleting the login, a copy of each system's password file is sent to the "hub" systems by deluser via "ins". If the "-y" option was also specified, sysadmin1's login directory would have been removed, if in doing so, no other login directory or system directory (e.g /usr)

would be removed. Alternatively, the login could have been disabled instead of removed by executing:

```
ins +all -n -x /pkg/pwadm/bin/chguser -o sysadmin1 -p void
```

This would change the password field of sysadmin1 to the string "void". No password could be crypted to this string since the password field consists of 13 characters. The example below further illustrates chguser's capabilities.

```
ins +sysfoo -n -x /pkg/pwadm/bin/chguser -o mylogin -l newlogin -d /fs1/newdir -s /bin/ksh
```

If "newlogin" already exists in "/etc/passwd" chguser would not effect the change and would print an error message. This command also changes the home directory to "/fs1/newdir", moving the contents of the old login directory to "newdir". Chguser ensures that moving the directory will not impact other users before actually moving the files to the new directory. It also verifies that the user is not currently logged in before proceeding.

The adduser command is very similar to the examples above except that more parameters are mandatory since each field in the password file entry must be specified. "Pwadm" also contains utilities used to move logins from one system to another, collect unused user ids, and perform password file validation.

"Pwadm" has been in use since January of 1985 and has continued to meet the needs of our organization. It has significantly reduced staff hours required to maintain logins and eliminated the corruption of password files caused by ad-hoc methods of implementing changes. It has also provided centralized control of the password files without requiring any modifications to standard Unix commands (e.g. /bin/passwd). Most importantly, it has become a strictly adhered to convention because of its ease of use and completeness.

Makealiases - a mail aliasing system

Gretchen Phillips - SUNY@Buffalo
Don Gworek - Sun Microsystems

September 1988

1. Introduction

The State University of New York at Buffalo has an ever growing Unix based computing environment. The primary instructional Unix machines now include a Vax 11/785 and a Sperry 7000/40 both running Berkeley 4.3 as well as an Encore Multimax running Umax 4.2 Release 3.1. Additionally, Sun workstations are beginning to proliferate on the campus. Our Unix users range from freshmen who have accounts so that they can "learn the system" to graduate students in Computer Science and Electrical and Computer Engineering who are primarily interested in operating systems and networking.

The total number of unique usernames is over 1000. This is broken down to about 500 undergraduates, 120 faculty and staff, 50 system logins, 25 support staff, 50 miscellaneous accounts and the remainder (about 300) graduate student accounts. Users are assigned accounts based on their particular needs and the ability of the machines to accommodate users. Users needing specific products (e.g. macsyma) are assigned to the machine with that resource. Users needing general access are assigned to the most lightly loaded machine. This may change from semester to semester depending on what classes are offered and the research projects that are particularly active. Nonetheless, users still need to receive their mail in some reliable, consistent (and easy to maintain) fashion. To this end we have developed a program *makealiases*.

2. Philosophy of *makealiases*

When users have multiple accounts on many machines within a domain it is possible for them to receive mail on any (or all) of the machines. *Mail* provides the feature of a *.forward* file for mail forwarding. This is not a satisfactory method for mail delivery considering that nearly one third to one half of our Unix users are neophytes. They are primarily concerned with getting their programs edited and running and not with where their mail is being delivered. Additionally, it was possible for a student to have accounts on one or all machines depending on their class enrollments. This means that mail would be delivered in, what appears to them, some random manner.

We decided that it would be easiest to generate an aliasing system within the domain that would assign a single machine as a user's primary (and only) mail drop. All mail sent from within (or from outside) the domain would be delivered to one machine for each individual. The machines would be assigned a priority for

delivery for each group. This allowed faculty to receive their mail on the machine that they do their research and undergraduate to receive mail on a machine that they would be doing their class work. If a user has accounts on multiple machines this priority system decides where to deliver the mail. If a user only has a single account, then their mail is delivered to that machine.

One significant problem that this overcame was that the sender does not need to be concerned with knowing where an individual reads their mail, or even on what machines they have an account. All accounts within the domain can be addressed simply by the username. Perhaps this situation is a common one, but we are satisfied with our solution.

3. *makealiases* implementation

Makealiases updates and distributes /usr/lib/aliases to all machines in a local network. It provides a single mailing address for users with multiple accounts, and expands special group aliases that are local to a host. *Makealiases* should be run after new accounts are created, to keep /usr/lib/aliases up to date. *Makealiases* should be run on only ONE host in the network.

3.1. Format of *aliases* file

The postmaster makes changes to system mailing addresses in the file /usr/local/lib/mail/aliases. This is a copy of /usr/lib/aliases. The file *aliases* should be organized in four sections:

- (1) Distribution and priority information should be at the start of the file. Each of these lines begins with a '#', followed by the word **dist** or **priority**. For a **dist** line, the second word is **special**, **hosts** or **yphosts**. **special** is for hosts that do not need the aliases file to be installed and do not want the mail list. It merely gets a copy of the aliases file. **hosts** defines all hosts that are to get a copy of the final aliases list. Any hosts preceded by a ! does not have its /etc/passwd file sampled and therefore will not have any mail delivered to it. **yphosts** defines hosts that have yellow page listings that must be updated. **priority** lines define what order mailboxes are to be assigned depending on the prefix specified in the line. If a username is not prefixed by a priority prefix, then the **default** definition is used. This is the order in the **hosts** list if a **default** is not defined. Only hosts in a priority list are required to be up for a run. Information from other hosts are used only if they are up. This section is ended by the line

MULTI-USER ADDRESSES ###.

- (2) Multiuser aliases and special groups should be listed between the

MULTI-USER ADDRESSES ### and the ### FORWARDING ADDRESSES ### line. *Makealiases* expands the special groups for the specified host.

- (3) Forwarding addresses. Users who desire mail on a specific local host, or users who receive mail nonlocally, should be listed between the
`### FORWARDING ADDRESSES ###` and `### AUTOMATED_BEGIN ###` lines. This section of the file is optional.
- (4) Automatically generated individual aliases appear after the `### AUTOMATED_BEGIN ###` line. After this point, *makealiases* appends aliases for users who receive mail on remote machines.

3.2. Other related files

The postmaster can specify expansions for special groups local to a host in *specialgroups.host* files found in the group directory. Special group files are optional. These expansions usually include a local archive file name, or a pipe to a local program. In general these are addresses not desired on all hosts. The syntax and format of the expansion is the same as a normal mailing alias. Previous copies of */usr/lib/aliases* are located in the archive directory.

3.3. Running *makealiases*

When *makealiases* runs, it first checks that all required hosts are up. Then it forks off a child to run in the background, and the parent dies. The child collects all usernames from the hosts. And then each host is individually updated with a new aliases file and a new */usr/lib/mail/mail.local*. (This is a file listing all users, where their accounts are, and where their mailbox is located.) After all hosts are updated *makealiases* sends a letter to the user of the program, to the postmaster, and, locally, to a special group *makealiases*. This is a mail-record file of all runs of *makealiases*. The letter contains a transcript of the run, including the differences that were found between */usr/lib/aliases* and the seed *aliases* file, and the output from the *newaliases*. *Rmakealiases* is a shell script run by cron once a week to update */usr/lib/aliases* and */usr/lib/mail/mail.local* on nonequivalent hosts (locally, our Sun workstations). *Cronmakealiases* is a shell script that is run nightly by cron. If there is a difference between */usr/lib/aliases* and */usr/local/lib/mail/aliases* it then runs *makealiases*. With *cronmakealiases*, changes to aliases can be made in the daytime, and *makealiases* run at night when load is low.

A Notice Capability for UNIX

Michael A. Erlinger, Ph.D.

Computer Science Department, Harvey Mudd College

Introduction

Harvey Mudd College is the engineering and science undergraduate college of the 6 member Claremont Colleges. At Harvey Mudd the Computer Science Department runs a small UNIX® network of 2 machines: muddcs, a VAX 11/750 and jarthur, a Sequent 21K multiprocessor. The machines are used mainly for class work and small research projects and currently have about 500 user id's. The system is administered by students under the direction of a faculty member. Historically, the systems have been limited on the amount of available disk space.

The Problem

We discovered that one of the most critical uses of the system was to communicate to the students within each class and to the members of various projects. Mail was not satisfactory because it utilized too much disk space; i.e., the copies of each message held for each addressee in the spool directory and the copy that large numbers of users kept in their own mail directory. (It seems that no amount of cajoling, chiding, etc., can make a vast quantity of users delete old mail messages.) Thus the problem was to develop a system that allowed a single copy of a message to be made available to the appropriate set of users. It was also determined that:

1. *the users should be shown the message rather than informed of the existence of a message.* We wanted to ensure that there was no excuse based on the standard, "I didn't see the single line informing me that I had a notice."
2. *a message should expire based on a date determined by its creator.* The message creator is the best person to determine how long a message should be available to any particular group of users.
3. *messages should actually be removed from the system by a system command.* We wanted to make sure that the system staff had control over actual deletion of the messages. Our reasoning was based on the fact that at the end of a semester numerous faculty had indicated a desire to review the notices pertaining to a particular course. Thus by keeping the notices within a library of notices even though they had expired and were unavailable to the users, we were able to assist faculty.

UNIX® is a registered trademark of AT&T.

The Solution

We create a group of programs to create notices, to review notices, to expire notices, and to delete notices. **notadd** allows a user who is in the *notice* group to create a notice. The *EDITOR* environment variable is used to put the user into the editor of his choice or into the default editor, *jove*. Once the editor is exited, the creator is prompted for the expiration date and the *groups* that are to receive the notice. When each user logs into the system a file in the notice directory is checked for all unexpired notices; then the group listing in each of the unexpired notices is compared to the groups of which the user is a member; finally those notices that are unexpired and for groups of which the user is a member are run through **more** as part of the login process. Besides creating **notadd** we had to modify **login** to search the notice directory, */usr/spool/notice*, and the kernel limitation on the number of groups to which a user could be a member had to be modified.

There is also a program, **notice**, which allows a user to review all the non-expired notices that pertain to him.

The two programs **notexp** and **notdel** are used to manage the library of notices. **notexp** is run daily (automatically) to remove from the list of active notices those notices that have expired, thus shortening the list of notices to be searched at login. **notdel** is run once or twice a year (semester break and summer) to remove all the expired notices.

Summary

Besides the use for particular classes and projects, the notice system has proven to be a very useful addition to daily system management. Rather than keep modifying the *motd*, we use notice to inform all users of down time, system changes, lectures, talks, etc. Notice was one of our first system programs and illustrates the majority of useful system management tools: it is simple, it is easy to maintain, and it did not require a major development effort.

A Batching System for Heterogeneous Unix Environments

Helen E. Harrison

Microelectronics Center of North Carolina
heh@mcnc.org

It is well known that jobs which require a large amount of virtual memory will run faster if run sequentially because reduction in the overhead of context switching and, more importantly, in paging. In addition, in our environment, users often have many jobs that they would like to run at off hours, when our computers are less heavily loaded. Rudimentary batching can be done using the at command, but this provides no scheduling coordination to maximize cpu cycles. The MCNC batch command allows users to submit jobs for sequential background processing. Batch will submit jobs to queues either on the local machine or to any remote machine running the batching system. The user does not have to have an account on the remote machine. This allows us to use a machine primarily as a batch processor, without the overhead of maintaining interactive user accounts. The batching system currently runs on 4.3 BSD and Convex (4.2 BSD based) UNIX*. Jobs are actually run by a daemon normally started at boot time. The flow control and networking routines were largely lifted from the Berkeley lpr/lpd line printer spooler. The user commands functions similarly to the lpr utilities.

The batch command puts the requested job in one of several available queues: short, medium, long, or overnight. The queues are distinguished by time limit on the jobs. A 'short' job must finish in less than 15 minutes of cpu time; a 'long' job has no time limit. Jobs from the long queues are run at a lower priority (niced) than those from the short queues.

When a job is submitted to run on a remote machine, batch copies any input files to a temporary directory on that machine. When the command completes it returns to the user any output files, optionally including a file which contains the standard output and standard error from the run. A command line option specifies that mail be sent to the user when the job has completed. The command to be batched must be one of a set of commands listed in the file /etc/jobcap on the host requested to run the command. This file also lists which users may run this command; if no list is present, anyone may run it. In addition, jobcap specifies a default host on which to run each command, if no host is specified on the command line.

A typical command might look like this: *batch -r remotehost -q overnight -i infile -o outfile -c bigprog -xyz -i infile*. This command submits to the *overnight* queue on *remotehost* the command "*bigprog -xyz -i infile*". Batch copies *infile* to

*UNIX is a Trademark of AT&T Bell Laboratories.

remotehost and expects to find *outfile* when the command finishes. *outfile* is returned to the local directory from which **batch** was called. **Batch** does no processing of *bigprog* or its options. Therefore it is necessary to tell **batch** explicitly what the input and output files will be. Instead of the -o option, one may specify an output directory with the -d option. In this case **batch** copies all files in the temporary execution directory which have been modified to the requested output directory. This is useful if there are too many files to list conveniently on the command line. In addition one may specify an empty file which should be created when the job finished. This is useful if jobs are being batched in an automated way. For example, one could write a shell script which waited and tested for this completion file, and then submitted another job if the output indicated that the previous one had been successful.

The batch daemon, **batchd** uses the Berkeley socket calls `listen(2)` and `accept(2)` to communicate with daemons on remote machines. When invoked, **batchd** reads the file `/etc/batchcap` to find out which queues are available on the local machine. This may vary from one machine to another within a batching system. **Batchd** will accept remote jobs for any of its local queues from any one of the machines listed in the file `/etc/hosts.batch`.

Two other utilities are associated with the batching system: **batchq** and **batchrm**. They work much like their **lpr** counterparts. **Batchq** reports the status and contents of the specified queue. **Batchrm** allows a user to remove a currently queued job if he has sufficient permission to do so. Both of these commands, like **batch** itself, can be asked to operate on queues on remote machines. In summary, while timesharing systems are more generally useful, sometimes a batching system is more efficient. Our system works well and we use it heavily.

Lazy Man's Guide to UNIX System Administration

Bjorn Satdeva

uunet!sysadm!bjorn
/sys/admin, inc.
(408) 353 3744

The proposed presentation will be about tools which can help the Unix system administrator in his/her daily work. Although most of the tools currently are in a development phase, the design are based on experience gained from a number of shell scripts and small C-programs, written in order to off-load the system administrator of his daily work.

The main goal behind the design is to off-load the system administrator of many of the small and irritating tasks which has to be performed every day, as e.g. checking free disk space, maintaining passwd files and mailrc files, etc. Each of these tasks is either taken over by automagical utilities, or is delegated to the users on each machine in a safe manner. None of the utilities in presentation will be aimed on installation and configuration of machines, but is rather a way to distribute tasks to the user community, without sacrificing security and reliability of the systems.

The tools now being developed differ from the original scripts by being faster, reliable, and secure. The first generation tools could only be used in a reasonably trusted enviroment. Also, the second generation design, implemented entirely in C. Many ideas which were not practically implementable in the first version can now be included.

The main goal of the presentation will be to share the experience from the first generation of tools, and to outline how we see new system administration tools should be designed.

Netdump: A Tool for Dumping Filesystems

D. Ryan Hawley

Sun Federal Microsystems Inc.

This is the man page for the main script I use to do network backups at night, then I use dump(8) to back up the compressed "netdump(8)" files. This saves time, and tapes.

NETDUMP(8) MAINTENANCE COMMANDS NETDUMP(8)

NAME

netdump - incremental (daily) dump

SYNOPSIS

netdump filesystem...

DESCRIPTION

Netdump create an full (level 0) dump, pipe the output through compress(1), and store the result in a file in an NFS mounted directory, for example: /usr/Big_Dump or /usr/Lil_Dump. This will be site dependent, it depends on the size of the partitions you will back up, and the size of the NFS partitions you will be dumping to. The file name is the same as the name of the file system(s) being dumped

FILES

"dumpserver":/usr/"DUMPDIR"/"unique_partition_name" For example: dumpserver:/usr/Big_Dump/xy0d

SEE ALSO

dump(8), restore(8), compress(1), netrest(8)

Extended SunOS Release 3.5 Last change: 8 September 1988

Combining Two Printing Systems Under a Common User Interface

Dave Goldberg

Bedford Computer Center
MITRE Corporation

Introduction

The Bedford Computer Center (BCC) at The MITRE Corporation maintains a wide variety of computers, operating systems and printers. In 1985, the BCC developed the Distributed Printing System (Dprint) to provide a common set of printers to users at MITRE-Bedford, which comprises more than ten buildings, and MITRE-Washington in McLean, Virginia.

Now that PostScript printers are being connected to the local Ethernet, the BCC has to work the new printers into the Dprint System. This has proven to be a difficult task on Unix, because the PostScript printers must be accessed with the Berkeley Line Printer Spooler. The problem is to combine the two printing systems under a common user interface.

The Distributed Printing System

The Dprint system is made up of an IBM 4381 running TSO connected to a set of Xerox 2700's, IBM 6670's and a Xerox 8700. There is printing software for the 4381 and packages on the other systems that send print jobs to the 4381 over Comboards (with the exception of a 4341 running VM, which has a direct SNA connection). The Dprint software was written locally and designed to provide as common a user interface as possible across the different operating systems at MITRE.

The user interface for the Dprint System as it exists on the BCC Ultrix system (mbunix), consists of two commands: dprint and mprint. The dprint command accepts command line options specifying the destination printer and various printer options such as font, full page bolding, landscape/portrait and margin widths. These options can also be specified in a file named .DPRINT in the user's home directory. The dprint command uses these options to create a JCL header. It then appends the standard input, or files specified on the command line to the JCL, converts to EBCDIC, and sends the file to TSO over the Comboard. For those new users who have used Unix before, the lpr command is replaced with a call to dprint.

The mprint command is a menu-based front end to the dprint command created with the curses-based Maryland Windows package. Mprint makes it easier for users to take advantage of the myriad of dprint command line options without having to memorize them and their restrictions.

A New Printing System

The BCC is now in the process of moving to a variety of Imagen PostScript printers connected to the local Ethernet. These printers, which will replace the Xerox 2700's, will be accessed directly from mbunix and BCC VMS hosts. The IBM hosts will access these printers by sending jobs to one of the VMS hosts via a Comboard. The VMS hosts and mbunix will continue to access the

Xerox 8700 and IBM 6670's by submitting requests to TSO by way of Dprint. Jobs from mbunix will be sent to the Imagens via the Berkeley Line Printer Spooler (Lpr). Thus, there will be two separate printing systems on mbunix: Lpr and Dprint.

The Problem with Lpr and Dprint

The problem with the two printing systems is that they provide different user commands. The dprint command has a set of options that do not match those of lpr. While they use the same option letters in many cases, the letters have different meanings to the two commands. This conflict would be a source of confusion to Unix users, many of whom are novices. Although the mprint command, with its menu interface, will alleviate the confusion in many cases, it does not accept standard input, and so does not provide a complete solution.

The Initial Approach and Its Shortcomings

The initial solution proposed was to simply modify the dprint command to call the lpr command, which would be hidden away in a directory not normally accessible to users. This proposal presents a serious problem, however. To enable users to send jobs to the Imagens, the BCC purchased the Transcript Package from Adobe. Two commands, enscript and ptoff, make calls to lpr to send the PostScript they create to a printer. Because the BCC purchased source to Transcript, we could modify enscript and ptoff; but that would mean having to modify them every time Transcript was updated. That was considered undesirable. It became clear that equating the dprint and lpr commands would not work because of the conflicting command line options.

The Final Solution

The final solution, currently being developed, is to combine the two systems under a common command and provide the menu interface. The new command will be callable either as dprint or lpr. These may be used interchangeably with two exceptions. The first exception is that if the user calls the program as lpr, then the destination must be specified with the -P option; if called as dprint, then the -d option must be used. This restriction is necessary, because both options have different meanings under the two commands. The second exception is the use of the -h option under lpr. Because we have so many users printing to the same printer, we are not allowing the job header page to be suppressed. The -h option under lpr will be treated the same as it was under dprint: it prints a short help message followed by the user's defaults. Any other command line options will be either dprint or lpr options depending on the destination printer. This lets a user invoke the dprint command to print to any printer, and the enscript and ptoff commands to call lpr.

At first glance this does not appear to solve anything. There are still two sets of command line options. However, in the case of the PostScript printers, there is only one useful option: the destination. MITRE in general does not encourage use of troff or other such packages. Therefore, users will rarely, if ever, have to call for a filter other than the default text filter. All most users need to remember is that the Imagens will not accept dprint options other than -d to specify the desired printer. They will not be required to learn any lpr options.

On the other hand, those users who do use troff will most likely be printing with the ptoff command, so they need only be concerned with learning the syntax for ptoff. For employees with Unix experience outside MITRE, they will be able to use the lpr command to print to any printer they desire, including the 8700 and 6670. The user will just have to be aware that these printers do not have drivers for troff or other packages, and therefore can only be used for text.

Implementation

The system is implemented as a C program that checks argv[0] for the calling name (dprint or lpr), and determines the destination printer by checking the command line for a -P or -d. If no such option exists, it checks the environment variable PRINTER. If that doesn't exist, it tries a .DPRINT file in the user's home directory, and finally, if all that fails, it defaults to the 8700. The default will probably change to a DEC Printserver 40 when DEC provides Ultrix support for it. The program then creates a command line for the "real" lpr or "real" dprint, both of which are hidden in a directory not normally accessed by users, depending on the destination. All options from the current command line are passed to the command. If the desired files are specified on the command line, then the system routine is used to execute the command. If the program is the tail end of a pipe, then it uses popen to open the command and writes the standard input to it.

The mprint command will be modified as well. It currently pops up menu choices based on the type of printer. For example, the IBM 6670 supports vertical spacing (i.e., single space, double space) but the 8700 does not. Therefore, the choice for vertical spacing is only presented if the destination printer is one of the 6670's. A new set of menus will be created for the Imagens. Unlike the Xerox 8700 and IBM 6670, the Imagen printers' capabilities will not be defined by escape codes. Rather, appropriate PostScript must be created to be sent to the printers. Therefore, the menus will provide an interface to the options of the enscript command, which formats text files. The option to send a raw PostScript will be provided as well.

Other Issues

There are two other issues involved with the printing system: printer accounting and printing from other Unix hosts that do not have a Comboard connection to TSO. Accounting of jobs sent to the 8700 or 6670's will continue to be handled by the TSO machine. Accounting for the Imagens will be handled on mbunix with accounting software provided by Imagen, slightly modified to use local conventions for project numbers.

Printing from remote Unix systems is currently handled by creating a JCL file on the remote machine and converting to EBCDIC. Then a daemon checks for complete print files, and if it finds any, rcp's them to mbunix, then uses rsh to tell mbunix to send them to TSO. This scheme will remain for printing to the 8700 and 6670's, but since the Imagens are connected to the Ethernet, the Imagen software can be installed on the remote systems so they can access the PostScript printers directly.

Conclusion

The effort to provide PostScript to MITRE computer users requires a great deal of change to Dprint. In some cases, the PostScript printers have been incorporated relatively easily into Dprint. On Unix, however, the change was more drastic because it required use of the Berkeley Line Printer Spooler. By combining the two printing systems under a single command set, users can access the PostScript Imagens without having to learn a new user interface.

A Flexible Backup System for Large Disk Farms

or

What to do with 20 Gigabytes

Helen E. Harrison

Microelectronics Center of North Carolina
heh@mcnc.org

Backing up 20 gigabytes disk space is challenging in the best of circumstances. The varying uses of our machines require different levels of backups. We have developed a backup system which is flexible enough to accommodate different needs while remaining manageable. The MCNC computer facilities include 2 VAX 8650's running 4.3 BSD UNIX*, a VAX 11/750 running 4.3, 2 Convex C-1's (4.2 variant), a MegaOne chip tester (4.2 variant), an assortment of Microvax II's and III's running 4.3, approximately 25 DEC Vaxstations running Ultrix 2.2, and a couple of Sun workstations. We expect the number of workstations to continue to increase in the coming year. MCNC has a full time System Support staff of 6 programmers, 3 operators, and a technician.

In order for our backups to be manageable, it is important to make the procedures look the same across all different types of machines, and make them easy to perform. First, it is helpful to make the machines themselves look as much alike as possible. One way in which we do this is use of symbolic disk and tape names. All filesystems are referenced by a mnemonic name rather than the physical partition name. It is much easier for operators (and system programmers) to deal with filesystems as /dev/usr or /dev/mcnc instead of /dev/ra7h or /dev/hp3f. This has several benefits. When the physical location of a filesystem changes we need only change symbolic link for that filesystem. Different vendors have different names for various tape drives (rmt8, rst0, rmt0h etc). We also create symbolic names for all tape drives so that /dev/rmt0.1600 is the lowest density for the first tape drive (of any type), and /dev/rmt1.6250.norewind is the high density, norewind specification for the second drive, etc. This hides the specific details about each machine, and provides a consistent user interface.

While our naming scheme made things easier, we still had the problem of keeping up with 195 filesystems. To solve this we created a single 'database' with entries for each machine, which filesystems to backup, and which tape drives are available to that machine. All of the various backup scripts use this database to determine what to do or what to use as defaults.

From the viewpoint of backups, each of our machines is a standalone computer. Backups are run on each individual machine using the standard Berkeley

*UNIX is a Trademark of AT&T Bell Laboratories.

dump utility. The basic backup scheme involves 3 levels of dumps: a monthly full backup (level 0), a weekly intermediate dump (level 2), and a daily incremental dump (level 9). The level 0 and level 2 are done interactively. Nightly level 9 dumps are done automatically. In addition, some important machines also have level 9 dumps each morning. The interactive backup scripts present the operators with menus of defaults from the database, each of which may be over-ridden as necessary. The disk permissions are set to be readable by a privileged group and the dumps are done as the user 'backup' who is a member of this group (i.e. dump don't have to be done as root).

Now that we have equalized the differences in the how filesystems look across machines, I should say that some filesystems are more equal than others. On our 'big machines' it is unacceptable to be without a filesystem for any extended period of time. To prevent this, we have on-line backups of these filesystems, complete copies which are created each night (via 'dd'). Our 'most important' machines also tend to be the busiest machines. These read-only backup filesystems allow us to have inactive filesystems that can be dumped during the day. This not only minimizes down-time, but also allows users to retrieve their own recently lost files. Workstation dumps are done a little differently. The automatic evening and weekly dumps for each machine are sent to a file on the central disk server. (Thank you, Project Athena.) These dumps are then backed up as a matter of course with the server's filesystems. Likewise, monthly dumps of each workstation are done from the server onto the server's 6250bpi tape drive. In summary, we use a variety of backup tactics, each addressing a particular need, and providing an excellent mix of access to old files and ease of administration.

Andrew Backup System

Stephen Hecht

Carnegie Mellon University

Introduction

One of the design goals of the Andrew File System (AFS) is to provide personal computer users with many of the services that were available to them on large timesharing systems. Users have come to expect transparent file access, the ability to share and protect files, behind-the-scenes file system administration, and centrally controlled file backups on the Andrew File System.

The Andrew File System allows UNIX workstation users to have access to a centrally administered file system. AFS is much like the UNIX file system in that it provides a hierarchical file system structure consisting of disk partitions, directories, and files. However, unlike UNIX, AFS provides an additional layer between the file system and the disk partition. This layer breaks the partition into many mini-file system chunks called "volumes".

Requirements of the Backup System

- All read-write volumes must be backed up, except for temporary testing volumes. Currently, the total size of all read-write volumes in the campus-wide file system is 14 gigabytes.
- File server performance may only suffer during off-peak hours.
- The backup system must be fairly resilient to file server downtime: backing up a volume on an unreachable server should not be completely skipped, but rather re-tried periodically.
- Interaction with operators should be kept to a minimum. Emphasis is to be placed on the autonomy of the system.
- The number of tapes mounted when backing up or restoring should be minimized as much as is practical, in order to reduce the load on the operators.
- The behavior of the system must be tunable. All operations should be parameterized, with adjustments taking effect as soon as possible.

Limitations of Existing Backup Software

- The 4.3 BSD backup utilities, dump(8) and restore(8), do not adequately handle Andrew backups.
- Although dump(8) can backup a volume (considering it to be a file system), the volume dump created would not contain the vital hierarchical information needed by restore(8).
- Operators would be required to have a detailed knowledge of the system. They would be responsible for selecting the correct tapes for backups and restores, maintaining backup location databases, monitoring tape usage and recycling, and performing any cleanup operations.
- For maximum efficiency, concurrent dumping must be performed. However, this would require a tape drive to be attached to each file server.
- The system would be difficult to adjust, and its organization would depend too heavily on operators.

Implementation

The current backup system runs on a VAX 780, which has two TU78 tape drives attached, one used for storing backups, the other for restoration (although one tape drive would suffice). Hard disks are used as a buffering area for backup dumps. The size of the buffering area is in proportion to the size of the file system being backed up, and the frequency of backups. Currently 8 Eagle disk drives are used, providing roughly 2 gigabytes of buffering space. The backup machine for a smaller file system is currently using only 1 CDC drive with 650 megabytes of disk space.

A volume is the smallest unit available for backup and restore. All operations consist (at some level) of manipulating dumps of single volumes, although the interface often operates on groups of volumes.

During off-peak hours, volumes are copied from the Andrew file servers into the buffering area of the backup machine. These transfers can take place concurrently, and often one or more volume dumps are being received from each file server simultaneously.

During the daytime, the volume dumps which have accumulated on the disk drives are sorted, and appended to magnetic tapes. By default, the tape selection algorithm attempts to write as much on one tape as possible, in order to minimize the number of tapes mounted.

Restoring volumes is done by system administrators, upon request. The volume and date desired are given to the backup system, which retrieves the proper full and incremental dumps from backup tapes, merges them, and sends the resultant volume to a file server, either as a separate read-only volume, or as a replacement for a corrupted volume. File server partition restores are also available, and are

fairly efficient since volume dumps are sorted by file server and partition (among other things) before they are written to tape.

The system requires virtually no training of the operators. Its only interactions with them are (1) Requests for tape mounts, and (2) Requests for new tapes to be added to the system: a new tape must be labelled, and a line must be added to a text file identifying the new tape to the system.

The system is controlled by a set of configuration files, which specify:

- the distribution of volumes into groups
- the frequency with which volumes in different groups are backed up
- how long volume dumps are kept before being deleted
- how full the buffering disks may become during dumping
- what servers/volumes are to be backed up
- the partitioning of tapes into groups (corresponding roughly to volume groups)

Most operations are initiated by cron(8): nightly volume dumping from the servers, staggered tape mounting requests during the daytime, and periodic checking for restore requests. However, changes to the configuration files are monitored by a daemon process, which updates the system accordingly.

Performance

Currently, about 2 gigabytes of data is transferred nightly from the file servers to the backup machine. The campus-wide file system contains just over 8000 read-write volumes. The backups system tape library consists of about 1100 tapes (of which approximately 100 are blank due to normal recycling operations).

Current schedule dictates full dumps of user volumes once per week, incremental dumps every other night (due to the vast number of user volumes: about 7600). All other volumes are completely dumped once every two weeks, with incremental dumps every night.

Incremental dumps are deleted from the system after a month, full dumps after several months (depending on the type of volume; end-of-semester archives are never deleted). Sorting done during tape-appending causes most or all of the dumps on a given tape to expire at the same time, making for efficient recycling.

Largest limiting factors are currently (1) speed of TCP, (2) operator delay in mounting tapes, and (3) to a smaller degree, size of buffering area. Database operations and sorting/selection algorithms have been optimized where inefficiency was detrimental.

The system is fairly autonomous, and requires almost no administrative intervention during normal operation.

Plans

Currently, the backup system is almost completely detached from the remainder of the Andrew File System. Better integration is necessary in order to share common databases, such as volume- and server-related information.

Currently, an administrator must be logged on to the backup machine to restore volumes, monitor performance, and adjust system behavior. Remote administration and redirection of diagnostic and statistical output would allow such tasks to be performed from any Andrew workstation.

Users should be allowed access to information about their backup histories, and to request retrievals. Such routine restores should be performed by the system, without administrative intervention.

BUMP - The BRL/USNA Migration Project

Mike Muuss, BRL

Terry Slattery, USNA

Don Merritt, BRL

Computer systems running UNIX range from personal computers to supercomputers. On systems with many users, or systems which run very large problems, disk space management can be particularly difficult. Space management has generally been accomplished by scripts and programs for determining "disk hogs" and mechanisms for users to explicitly move their files to some off-line storage media (magnetic tape, removeable disks, etc.), often using manual procedures and record keeping. In addition, UNIX programs attempting to write to a full file systems get an ENOSPC write error: "No space on device". Frequently, this behavior is not acceptable, especially in the supercomputer realm where programs may execute for a long time, after which there may not be enough space to write all the results to disk.

This paper reports on the implementation of a solution to both these aspects of the disk space management problem, by providing a transparent file migration and archiving facility. The result of this software is a the appearance of a UNIX filesystem that has significantly more capacity than then underlying disk drives, freeing the user community from worrying about managing off-line storage media.

The system administrator or CRON script may run a utility to cause certain files to be 'migrated' to one of several levels of offline storage. The I-node for each migrated file remains present in the filesystem, with special "file handle" data used to recover the file upon subsequent access. When a migrated file is opened, the kernel will block the user process, wait for the daemon to recover the file from backing storage, and then allow the user process to continue. This mechanism is entirely transparent to the user, except for the delay.

In addition, when a process attempts to write onto a full file system, the system can be configured to block the process, start a "space management" migration function to create file system space, after which the blocked processes will resume.

The details of the kernel modifications, support daemons, and related software necessary to provide fully transparent file migration will be presented. In addition, the software described is Public Domain, Distribution Unlimited. Several vendors have already expressed plans to incorporate it into their products.

A Simple Incremental File Backup System

Patricia E. Parseghian

Department of Computer Science
Princeton University
Princeton, NJ 08544
`princeton!pep`

In 1984 the computing facility at the Computer Science Department at Princeton University included six DEC VAX 11/750 computers running the 4.1 BSD UNIX† system. The peripherals included many large capacity, unreliable discs—and no tape drive! Inspired by an automated procedure run by colleagues at Bell Laboratories (Andy Tannenbaum, Brian Redman, Bill Hoyt), I developed a set of shell scripts to back up data incrementally to a removable disc pack on one of the 11/750s.

Today the department includes about 35 faculty and staff members, along with 60 graduate students and research associates. One ethernet network serves the department; a DEC MicroVAX II serves as our gateway to a campus wide-area network, the ARPANET, and external sites. The department's central computer is a DEC VAX 8650 with 60 megabytes of main memory and about 4 gigabytes of magnetic disc storage, running UNIX 4.3 BSD+NFS. Virtually all user files are there; each user has one home directory, whether it is seen from a discless workstation or a terminal attached to the 8650. We have about 40 workstations, most of which are discless SUNs. To the extent permitted by licenses and portability, all software runs on all CPUs (e.g., `troff`, etc.). The incremental backup scripts still form the core of our backup strategy.

Backing Up

The basic example is a machine with its own removable disc pack, such as our VAX 8650. Once each day, the scripts find all the files that have changed since the last time they ran successfully. The changed files are copied directly to a filesystem (`/save`) on the removable pack; specific files can be ignored by naming them in an `except` list and/or by setting a file size threshold. This limit is 5 megabytes on the 8650; files larger than that are presumed to be disposable, as they probably were created by a program (and can be recreated). A separate file tree is created on `/save` for each day's backups—older copies are not overwritten. The scripts also create an index of the saved files, which is stored online but *not* on the removable pack.

When the pack fills up (every 7 - 10 days), I take a full tape dump of the 8650's filesystems, including `/save`. Currently, we rotate through five disc packs. We rotate through four sets of full dump tapes for regular filesystems, but we retain the dumps of `/save` forever.

Restoring

Most users lose data they worked on recently, and chances are the files will be found in `/save`. We restore files for users; there hasn't been much demand, but it would be easy enough to let them restore their own files (if they're on the current pack, of course). If a file is not on the current pack, we search the index to identify the version we need to restore. This allows us to pick up one disc pack or set of tapes containing the file, and extract it.

If we lose a disc, I need only restore files from one set of tapes (the most recent full dump); the rest can be copied directly from the current incremental disc pack.

† UNIX is a Trademark of Bell Laboratories.

Backing Up Across Machines

We run the same scripts on machines that do not have removable disc packs. Files are spooled (via `uucp`) and sent to a DEC VAX 11/750 with a removable pack. When that machine's pack fills, I dump it to tape and rotate as above. It is also wise to take full dumps of the spooling machines at this time. Traffic is much lighter on these machines, so the central pack tends to fill up once every three months (even though it accepts backups from six or more machines).

Administration

Three Bourne shell scripts (totaling 130 lines) coordinate the backup. A fourth file contains eight parameters that are adjusted to match local circumstances (path names, file size limit, administrative logins for mail, etc.) A fifth file is a list of directory and file names that should not be backed up incrementally (e.g., ever-changing log files, USENET articles).

In addition to the 8650, the scripts are used on a DEC VAX 11/785 running Mach, two SUN fileservers, an Iris workstation (which runs a variant of UNIX System V), and two DEC MicroVAX IIs (one running Ultrix 2.0 and one running UNIX 4.3+NFS). The scripts have also handled backups for a DEC VAX 11/750 running the Eighth Edition UNIX system.

Performance

On our 8650, about 2.7 gigabytes of disc storage is scanned for changes each night; on the average, 1500 files containing 25 megabytes of data are copied to `/save`. The backup scripts are scheduled to run at 3 a.m., and the process is usually completed within 40 minutes.

When a user loses a recently-changed file, it can usually be restored minutes after a staff member is alerted (much to the user's relief!). Restoring an older file is somewhat more involved, but generally requires less than 20 minutes' worth of work.

Problems and Future Plans

There are two major shortcomings in the current approach, both related to dependence on `find`. If it is necessary to recover an entire disc (or a large subtree), it is easy to overflow the filesystem by restoring files that were backed up but later removed by users. Also, users are often confused when old files reappear (and annoyed that they have to remove them again). An elegant solution to this problem will make it practical to reduce the frequency of full tape dumps. The second problem is that Berkeley's `find` doesn't capture inode changes—such as file or directory renames, ownership or permission changes, or links. This is also a nuisance for users. Adding an option to `find` that picks up such changes is trivial, but I can't report on the effects yet.

One simple enhancement would be a tool that allows users to try to restore their own lost files. Another would be support for granularity finer than one day; the current structure of `/save` stores only the latest version of a file if the backup procedures are run more than once on a given day.

I have considered replacing the removable magnetic disc with a write-once optical disc. I am not convinced that this would be a wise choice for archival storage at this time. The optical media may endure for 30 years, but will I have the hardware and software to read it in 2 years? I have also thought about writing a server that would transfer files across the network without using `uucp` as an intermediary for inter-machine backups. However, `uucp`'s reliability and spooling properties get the job done well enough.

Backup at Ohio State

Elizabeth Zwicky

Ohio State University

As a university site with a lot of disk and not very much money, we have had to come up with a reliable way of backing up a lot of disk from several different sorts of UNIX systems to half-inch magnetic tape, using student employees, without buying commercial software. The system we have evolved uses fear, redundancy, and custom programming, and has now kept us from losing data that should have been saved for over a year (during which time we have lost 4 disk drives).

The machines we backup include 15 Sun 3/180 servers supporting 180-some Sun 3/50 workstations, 2 Sun 4s, 5 Pyramids (in an assortment of sizes), an Encore Multimax, a BBN Butterfly, 14 Hewlett Packard workstations, and 10 IBM RTs. All of these machines run one version or another of UNIX. None of the workstations have local disks. Between them they have 4 quarter-inch tape drives (which take 2 different tape formats), 2 half-inch tape drives capable of 1600 or 3200 bpi, 2 half-inch tape drives capable of 800 or 1600 bpi, and 5 half-inch tape drives capable of 1600 or 6250 bpi. They also have approximately 14 Gigabytes of disk space, mostly in Fujitsu Eagles.

Our first decision was to make everything consistent. We decided that we would back everything up with dump to half-inch, 6250 bpi tapes. In order to this, we had to port dump, restore, and rmt (the network tape driver) to some systems. We also wrote a shell script called ‘‘backup’’ on each system. backup is essentially a prettier front-end for dump. Some arguments (like level of backup) it prompts for, and others (the names and order of the filesystems) are hard-coded into it. When possible, we write multiple filesystems on the same tape; the decisions about which are likely to fit at what levels of dump are also hardcoded into backup.

We use a relatively complex system for levels and times of dumps. We divide file systems into system, user, and archive categories. For system file systems, we do a level 0 (full) dump initially, followed by a level 9 (daily) dump Sunday-Friday (onto a different set of tapes each day). On the next three Saturdays we do level 2 (weekly) dumps (onto a different set of tapes each week), and on the fourth we start over with a level 0. Each set of tapes is recycled after the next lowest dump has been done, so that daily tapes are kept for a week, and weeklies for a month. All full saves are kept for 6 months, and then a set for each academic term is saved for another 6 months.

The user file systems are backed up on the same sort of system, except that they get two level 9 dumps a day, to reduce lost work if a disk crashes. The archive file systems get no daily backups at all. Some that are particularly inactive get fulls when they are changed, and nothing else; for these machines all full saves are saved for an entire year.

Obviously this system provides a certain amount of redundancy, in that a given file is probably on several tapes. This has proved sufficient to prevent most problems with lost or damaged tapes, but grim experience led us to also adopt a highly redundant system for ensuring that the backups

get done when they are supposed to. Dump automatically logs backups in a file called /etc/dumpdates, and there is a field provided in /etc/fstab for the frequency with which a file system should be backed up. A program which compares the data in the two files runs every day 1 hour before we close for the night, and reports any backups that appear to have been missed so that they can be redone. There is also a paper log that lists information that does not appear in dumpdates (the tape drive that the tape was written on, and the name of the operator who did the tape), and it is checked by hand in the middle of the day. If either log shows a missed backup, the backup is redone.

An equally redundant system of tape labelling and storage is employed in order to help assure that tapes are never written over inappropriately. Tape bands are labelled with color-coded labels where the color specifies the day of the week (or the week in the cycle), and the level of the dump. The label also has the day of the week (or the week in the cycle), the machine name, the names of the filesystems on the tape, or a "tape n of m" notation if the filesystem is on more than one tape, and the level of the dump. The same information is written on the back of the tape using a permanent overhead projection marker (these can be erased with alcohol, but won't come off in water). Finally, different levels are stored in different tape racks. The dailies and weeklies are sorted into sets by the machine name, and the sets are alphabetized. The full saves are kept in chronological order in a separate room from the rest of the tapes.

As a last protection against inappropriate re-use of tapes, there is one operator who is a tape librarian, and only that operator can release a tape that has already been used, which he does by putting a special label that says "REUSE" on it, and moving it to a special place. If there are no tapes marked for re-use, and someone needs a tape that is not already labelled and hanging in the correct place for the backup they are doing, they take a new tape out of its box.

Dailies and weeklies are done when machines are up, but at low usage times (early morning, or during dinner). The morning backups are started before the labs are open to students, in fact. Full saves are done Saturday nights, when the labs are closed, and the machines are brought into single-user mode. The operators who do backups are not expected to do any other work at the same time, although they do run as many backups as possible in parallel, starting them so that they are staggered. Furthermore, of the 25 operators who staff our labs, only 4 of them share the load of the backups during any given quarter. This is a small enough group to allow them to coordinate effectively with each other, while still large enough so that having one person out sick is not a tragedy.

We are presently looking into several ways to improve the system. Currently, we are in the middle of rewriting backup, dump, and rmt, to increase features in backup, and to add speed to dump and rmt. We are also preparing to switch to higher capacity storage devices (probably 8mm tape) which would allow us to automate the procedure to a much larger extent. This would also reduce the size of our tape library, which has passed 1,000 tapes and is heading rapidly for 2,000.

Transitioning Users to a Supported Environment

Earl W. Norwood III

Associate Systems Administrator
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto CA.
enorwood@hplabs.hp.com

The Computing Systems Center (CSC) of Hewlett-Packard Laboratories (HPL) is engaged in the research and development of software engineering environments, programming languages, database technology disk storage technology, and multi-media interface technologies.

The Technical Computing Environment of CSC consists of three HP9000/840 HP PA computers, two DEC Vax C11C/780's, 17 HP9000/300 workstations acting as print/file/mail/nfs servers, 3 Sun workstations, plus approximately 185 HP9000/300 workstations used by individual researchers. The Precision Architecture and VAX machines serve as general time-sharing machines and compute-servers. One of the VAXen is the mail/news hub for HPLabs. The VAXen run BSD 4.3. The other machines are ATT SysV/R2 compatible. All of these machines are connected to a local area network (LAN) via Ethernet. The systems administration group is directly responsible for administering all time-sharing and server machines. In addition, we are the resource to aid researchers in the administration of their UNIX workstations.

Due to a recent reorganization, we were faced with the problem of moving 135 UNIX workstations and users from a largely unsupported or self-supported environment to a highly supported environment. There were major issues that needed to be dealt with in order to move such a large community of users. For example, there were different computing needs such as scientific research versus computer science. There were workstations with various versions of the operating system and different hardware configurations. There were also different types of workstations users. Some users wanted a 'turn key' type of system and did not want to know much about their workstations while others were very well informed of the inner workings of their workstations.

The goal of the Technical Computing Department (TCD) was to establish a support plan for these unsupported workstation users. We needed to find out their computing needs and establish standard configurations for hardware and software. In order to accomplish this, TCD support liaisons were assigned to the various labs inside of CSC to establish a single point of contact for workstation support. We set a priority list of departments to transition into the supported community. The TCD support liaisons went out to the new departments, presented them with the support plan, and examined individual workstations for software and hardware configurations. The standard configurations were applied, system administration scripts and utilities were installed. System administration utilities and scripts used on these and all other TCD supported systems are:

- Cleanup scripts that run daily, weekly and monthly, to remove old files in /tmp, remove old core dumps, trim log files, and remove old log files.
- Automatic backups to local tape or across LAN to a disk on one of our Vaxen.
- Scripts for backup retrieval.
- Ninstall star to maintain host tables, printer capability files, and do occasional minor software updates. (See "SOFTWARE DISTRIBUTION IN A NETWORK ENVIRONMENT" submitted by Mike Rodriquez, for last year's (1987) workshop.)

After the liaisons were finished, the users were consulted about what had been done to their systems and who to contact for help.

When deciding how to transition all of these users and workstations, we had to establish some kind of logical way to divide them up into smaller groups. So, TCD decided to do one department at a time with approximately 15-20 users and workstations in each group. These departments would be doing similar types of work and therefore have similar needs. This has worked well and we have been able to do about one department per month. This success is impressive considering TCD, still has our original base of 150 users to support.

We discovered several things during this process. What we in TCD were used to supporting were computer science researchers. What we found was a whole new kind of workstation user: the scientific computing user. Some of these users did not know any UNIX at all and did not want to learn UNIX. Their jobs called for them to use the applications they ran on their workstations such as CAD/CAM to do PC layout, IC design, data storage technology, imaging peripherals, and thin film research.

During this transition, it became clear that this is a continuous process; you make initial changes, introduce people to the support scheme, and start supporting them. Also, the computing environment here in HP Labs is changing. In the last seven months there have been two releases of HP-UX, the introduction of NFS/diskless workstations, and an early-bird release of X11. Another problem is that some users need some real hand holding. My partner found herself teaching one user how to use EMACS while another user needed a special kernel built to handle their application. We also found users using applications we had never heard of and yet were expected to support.

The transition allowed our user community to do engineering work on their HP-UX workstations without doing daily administration. The workstations get updates of host tables and printer tables automatically. The engineers have a single point of contact for any workstation problems. In some cases this has saved as much as a week down time when a workstation had a hardware failure. This gave the users somewhere to go for consistent help with problems in their computing environment and has helped them increase productivity.

Acknowledgments:

Mike Rodriquez for the first two paragraphs.

Andrea Chenu, for guidance and encouragement.

Catie Airriess-Norwood, Karen Bradley, for without their help this paper would have never gotten finished.

Making a Large Network Reliable

Steve Simmons

Inland Sea Software, Ltd.

9353 Hidden Lake
Dexter, MI 48130
scs@lokkur.uucp

For the past two years I have been head sysop at Schlumberger Technologies CAD/CAM division. There we had a large (170 node) network of Sun workstations. We put into places some methods to make the network very very flexible and resilient. The particular requirements that made this an especially hard one are:

1. The file servers were always driven to capacity, both in response speed and fullness. We averaged over 90% full on all disks all the time.
2. Work assignments changed with little notice. People would move from group to group, workstation to workstation, project to project, division to division.
3. Almost all projects were inter-related. Therefore closing off the network into subareas was not possible. A typical use mounted nearly half the file systems, and many workstations mounted all of them.
4. Extremely high uptime was demanded by management, but very little could be provided in the way of financial assistance.

We handled things by making the network appear to be a standardized collection of machines where one could literally pick up something from one slot and move it to another. Scripts were written to initialize file servers, public workstations, and personal workstations.

Non-standard hardware or non-standard configurations were resisted when possible, encapsulated when required. The simple technique of making management aware of the full cost of adding "just one little thing" was remarkably effective. Cost was typically laid out in both dollars and project delays.

We found ways of making "redundant hardware" for almost every piece of the network. Some of it was beg borrow and steal, some was negotiated out of management. We had cases where a 500MB disk supporting nine clients would fail and we'd be back on line in less than two hours *including installing the new disk*.

Since internal connectivity was very important, we built the network topology as a doubly connected star to a Delni which we dubbed SunMain. This pseudo-main allowed eight connections from four ethernets, and we used a LanBridge to connect it to a DECNET main.

Devices (eg, printers) attached to the network were configured in such a way that a defective unit could be removed from the network and its queues redirected to another target without losing the items in the queues and without requiring the users to take any actions. Although we only used this technique for printers, I believe it would work perfectly well for network gateways, plotters, etc.

By the time of the conference I should be able to report on how well these techniques can be used in a vendor-heterogenous network as well.

Update on Systems Administration Standards

Steve Carter

Bell Communications Research

The purpose of this standard is to provide a common set of utility programs and, if needed, service interfaces for systems administrators to use that,

- ensure the ability to install Computer Operating Systems and applications software complementing provisions in the 1003.2 POSIX document,
- ensure the ability to install, configure and maintain realtime facilities such as high performance file systems in conjunction with the 1003.4 POSIX document,
- establish the ability to install, configure and maintain the security related functions of Computer Operating Systems such as user profiles and privileges in conjunction with the 1003.6 POSIX document,
- ensure the ability to configure and maintain Computer Operating System Environments, and
- over time, in conjunction with the 1003 Networking Group (PAR submitted and pending approval), provide for administration of distributed Computer Operating Systems.

The standard is to be used by system administrators and implementors of system administration software. It is intended to be a reference document and not a tutorial on the use of the services, the utilities, or the interrelationships between the utilities

Standards and Guidelines for Unix Workstation Installations

James Hayes

Advanced Micro Devices
Logic Products Division
Sunnyvale, California
hayes@amdcad.amd.com

ABSTRACT

The recent trend away from centralized computing toward a uniform distributed workstation environment poses new problems in the area of system management. Standards and guidelines must be developed for the setup and administration of UNIX[†] based workstations or they will crumble into a mass of wasted time, wasted disk space, outdated software and unhappy customers. Users desire common commands. Users desire common administration procedures. Users want things to work the same, no matter which workstation they use.

1. Introduction

As computing price/performance ratios drop many corporate executives may be seen running from the traditional "one company one computer" philosophy and embracing the new-and-improved "one user one workstation" concept. In doing so corporations reap the benefits of increased productivity, increased computing capacity and better utilization of CPU horsepower. If not properly planned, corporations also discover that distributed systems breed chaos.

Users reap the benefits of standardized systems, lightly loaded systems and small learning curves associated with switching workstations. For the corporate manager the gains are immediate and long term.

For the system manager new to distributed environments, the only immediate gain is gray hair. Careful planning of a workstation environment is crucial to the success of the environment as a whole and must be completed before any workstations arrive. Once the workstations become operational and users are given access,

[†]UNIX is a trademark of AT&T Bell Laboratories.
Copyright ©1988, Advanced Micro Devices. All rights reserved.
Special thanks to Carl Rigney for all the editing, comments and ideas.

the setup might as well be cast in cement. Changes now are measured in expensive downtime instead of reasonably cheap setup time.

Buried at the root of the distributed computing nightmare is inconsistency. Picture twelve workstations with twelve different password files, some have disks, some do not; some have knowledgeable superusers, some reckless; there are five printers, some of which work only on some systems. It is quite obvious even to the uninitiated that this situation is pure trouble.

This paper is geared toward those "lucky few" responsible for managing a large network of workstations. Specifically, workstations capable of using the Yellow Pages database system (YP) and sharing files over a network using the Network File System (NFS). At Advanced Micro Devices these systems include DEC VAXen, Sun workstations and IBM personal computers equipped with PC-NFS.

The philosophy behind this paper is simple: Avoid future problems by planning ahead, writing it down and following what was written. This is accomplished by:

- Providing setup guidelines for system administrators that will give users a homogeneous environment across different system architectures.
- Providing guidelines for system administrators which will insure each system is kept up to date with regard to installed commercial and user contributed software as well as updates to dynamic system configuration files such as printer and network databases.
- Providing common procedures for root access on systems.
- Providing common procedures for logging system changes.

2. Workstation Setup Guidelines

2.1. Servers and Slaves

Using YP, the following combinations of master and slave servers should be set up at AMD:

- 1) One master server in Sunnyvale. (Clients always use the server that responds first; since the AMD Texas facility is 105 milliseconds away from Sunnyvale, response to clients in Texas will come from the local Texas slave.)
- 2) One slave server in each subdomain (i.e. `spd_sd`, `spd_support`, `lcad_networking`).

VAX is a trademark of Digital Equipment Corporation.
PC-NFS is a trademark of Sun Microsystems Incorporated.

-
-
- 3) Additional slave servers for subdomains with more than 4 diskless clients. There should be one slave server per 4 additional clients in the same subdomain.

2.2. UID's and GID's

NFS references remote files by the user and group ID of the requestor. This may be troublesome when each system and subdomain has its own set of user and group ID's. They shouldn't. Period. All password file entries should reside on the YP master server in `/etc/passwd`.

- 1) Standard system logins (`root`, `sysdiag`, `ingres`, `src`, `sync`, `shutdown`, `reboot`, etc.) should occupy the first thousand UID numbers (0-999). Regular user ID's should begin with 1000.
- 2) Special logins (system managers, root privileged “`{user}su`” accounts) should be located in one contiguous location in the password file. These rather special logins must be lumped together so they are easily found and not forgotten.
- 3) Standard group names (`wheel`, `src`, `bin`, etc.) occupy GID's 0-499. Company wide groups should occupy GID numbers (500-999). Regular group ID's should start with 1000.
- 4) A single postmaster should maintain a **single** `/etc/aliases` to insure its accuracy and integrity. Using a single alias database effectively creates a company wide “name server” using YP. The long term direction will be toward a true company wide dedicated nameserver based on RFC 1035 and RFC 974.
- 5) **Every** user in the master password file should have an entry in the mail alias database pointing to a single server. This is an effort to have ONE central mailbox for each user.
- 6) A record of ALL changes should be recorded. (Please see next section.)

2.3. Logging it ALL

At some point in time every user will require a change to his/her account. It might be a group addition, a new mail alias, more storage on another filesystem or modifying the system to meet the user's needs.

Now then, how do we undo the changes when the user leaves the company? Currently we rummage around the system and if we're lucky find about 50% of what we installed or changed. This is not good system management practice. How do we know what to reinstall after an operating system upgrade if we don't know which system directories were changed and what was done to them?

The following should be strictly enforced:

- 1) No system directories should be changed. (Standard system configuration files are exempt.) A system directory will be changed only when all other options have been exhausted. The change should be added to the CHANGES.LOG file in the root directory answering the following questions:

Who did the modification? (login name)
When did the modification occur? (date)
What files were added/changed? (full paths)
What directories were they put in? (full paths)
Why were the changes made?
How may the change be undone?

- 2) Any changes to system configuration files should be made through the Revision Control System (RCS) using the RCS portal in the same directory, allowing audit trails and incremental file restoration. The user making the change should identify himself in the RCS log message.
- 3) Local additions to /usr/local (please see the next section) shall be logged in the file /usr/local/README answering the following questions in the comment section. (Comments are lines beginning with #):

Who did the modification/installation?
Who requested the installation?
When did the installation occur?
What version was installed?
(A brief description should be included as well.)

After the initial comments, the package name should follow the keyword PACKAGE and package version should follow the VERSION keyword. After the keywords, the files in the package should be listed and a description given for each.

For example:

```
#  
# Jim Hayes installed elm 2.1 on October 31st, 1988. Indra Singhal  
# (indra@amdcad.amd.com) requested the installation. The following  
# files were added to /usr/local/bin and /usr/local/lib.  
#  
PACKAGE The Elm Mail Handler  
VERSION 2.1  
#  
  
bin/elm      The menu driven elm program  
bin/answer    The elm quick "answer" program  
lib/elm.help  Help files used by elm.  
src/elm      The elm source tree  
#
```

Source code and large packages with hundreds of files should be listed by the directory where the binaries/source are located. Configuration or other important files should be mentioned as well.

- 4) Any files in /usr/local not logged in the /usr/local/README file will be deleted once a week. No warnings. (Cron will invoke a program that will consult the README file and remove all unreferenced files.)

2.4. Files in /usr/local

All software packages should reside in /usr/local and should adhere to the following guidelines:

- 1) Local additions should be put in /usr/local under the appropriate directory. [Either: src, bin, doc, adm, include, man, lib, or etc]. Packages with large libraries (like the IMAGEN print spooler and ADOBE Transcript typesetting package) should have their own subdirectories under the standard ones in /usr/local. For example, /usr/local/include/imagen or /usr/local/lib/transcript.
- 2) Non-standard manual pages go in the appropriate subdirectories of /usr/local/man. In the default .login file given to new users, the MANPATH environment variable should be set to point to /usr/local/man and /usr/man
- 3) /usr/local/* should be public to all servers (of similar architecture) in a sub-domain. (So they can be mounted via NFS allowing *identical* commands on all clients.)
- 4) Stripping installed binaries (via strip(1)) is not necessary unless space is critical.

- 5) A publically executable shell script should reside in `/usr/local` called `TYPES` containing information about the system architecture and disk server. It should set the following environment variables that may be examined by shell scripts at will:

```
MACHTYPE (sun4 | sun3 | vax | ibmrt | ibmpc | ibmps2)  
DISKHOST (the host where /usr/local resides)  
ARCHTYPE (sparc | 68020 | vax11 | 8086 | 80286 | 80386 | romp | 29000)  
OSTYPE (ultrix2.x | sunos3.x | sunos4.x | bsd4.x | mtxinu4.x | xenix)
```

Here is an example:

```
setenv MACHTYPE "sun4"  
setenv DISKHOST "crackle"  
setenv ARCHTYPE "sparc"  
setenv OSTYPE "sunos4.0"
```

Because of the rather ugly nature of UNIX disk partitioning `/usr/local` may fill up. At that time, source code should migrate to a filesystem with more room on the same host. The resulting location should be documented in `/CHANGES.LOG` and a symbolic link should point to it from `/usr/local/src`. Source code should not be split across file systems, it must be kept in one spot to avoid confusion.

2.5. Source Code Access

Different types of source code require different levels of protection. Public source should not be read protected, while licensed source should. In the middle is local application source, which should be less restricted than licensed source but more restricted than public source.

Several rules should be followed fo the protection of source code:

- 1) When the dust settles in `/usr/local/bin` and an installation is stable, the source code (if it exists) should be placed in `/usr/local/src/dir/program_name`, where `dir` is based on the location of the installed program (i.e. lib, bin, etc.) For example: `/usr/local/src/bin` and `/usr/local/src/lib`. If the product, package, or program is large and has bits and pieces in all the directories (like a windowing system for example) the source should be placed in `/usr/local/src/package_name`.
- 2) At the time a program is installed, all its object files should be removed (except for system software like the UNIX kernel) and ownership given to the `src` dummy login. The group given to the source should be either `srclic`, `srcapp`, or `srcpub`, (for licensed, application, or public source respectively.)

- 3) Access permissions for source code in `srclic` and `srcapp` should grant the owner read/write permission, the group read only permission and the world no privileges.
- 4) Access permissions for public (`srcpub`) source code should grant the owner read/write permission, the group read permission and the world read only permission.
- 5) Users needing write access to source code should use a special version of the `su` command that logs all source access requests.

2.6. File System Names

With the advent of network mountable file systems, keeping track of mounted directories along with their contents and originating machines can be a nightmare. Standard naming conventions solve most problems. The filesystem namespace should be organized as follows:

- 1) Filesystems should maintain their original name across **all** NFS mount points. Calling a filesystem `/jones` on one workstation and calling the same filesystem `/smith` on another workstation is welcoming confusion and disaster.
- 2) User filesystems should be named `/amd/machine_name`, where `machine_name` is the host where the disk is physically located. The `/amd` before the machine name marks the filesystem as one that contains user data.
- 3) Home directories for each user should reside in a flat directory just under the mount point for the user disks. For example: (on the machine named `galahad`) `/amd/galahad/robin`, `/amd/galahad/michael` and `/amd/galahad/chris`.
- 4) File systems devoted to special functions (i.e. development, test vectors, UNIX ports, etc.) should be called `/project/machine_name/major_project_name`. Minor project names will be subdirectories off the major filesystem name. For example (again, using `galahad` as the workstation name):

```
/project/galahad/vectors/avp1
/project/galahad/vectors/avp2
/project/galahad/telecom/ISDN
/project/galahad/telecom/ethernet
```

When a new project or group topic is created a dummy account should be created with the same name so the home directory may be reached without knowing the actual path of the project. (I.e. `cd ~ethernet`, `cat ~debugger/gram.y`) The dummy account should have a "*" password, a unique UID between 500 and 999 and a GID between 500 and 999. The comment field of the account should have the name and extension of the person responsible for the project.

2.7. Network Groups

The Yellow Pages let system administrators associate groups of users to hosts allowing machine by machine access control. This allows specified users access to certain workstations, while denying others. Take the "support" and "spd" groups for example. Neither wants the other using their computers. The astute administrator creates two network groups (*netgroups*) called support and spd. The line +@spd is added to the spd group's password file and +@support is added to the support group's password file. Bingo. Marketing can't use the support departments computer and vice versa. (Similarly, the -@*netgroup* convention disallows members of a netgroup machine access.)

Netgroup definitions are stored in one file on the master YP server, /etc/netgroup, so naming them is important.

The netgroup namespace should be organized as follows:

- 1) Each subdomain (spd, support, etc.) should start with five default netgroups (where *xxx* is the subdomain name):

<i>xxx_hosts</i>	All the hosts in the subdomain
<i>xxx_hosts.equiv</i>	All trusted hosts in the subdomain [†]
<i>xxx_sysadmin</i>	The local sysadm "su" accounts.
<i>xxx_guests</i>	Any guests in the subdomain
<i>xxx_users</i>	The actual users in the subdomain

- 2) The netgroup consisting of special logins such as shutdown and sync should be put into the *special_logins* netgroup.
- 3) The netgroup consisting of AMD "unix expert" accounts should be called *amd_sysadmin*.
- 4) Local subdomain netgroup names should look like *domain_localgroup*. For example: *support_northern*, *support_western* and *support_offshore*.

2.8. Special Logins

The following special logins should be available on all machines as the *special_logins* network group:

No password:

<i>sync</i>	Synchronize memory buffers with disk buffers.
-------------	---

[†]A user logging in from a trusted host will not be prompted for a password, provided the same login name appears on both hosts.

With password:

sysdiag	System diagnostics
shutdown	Shutdown system giving monitor prompt
reboot	Shutdown system and reboot
network	Unprivileged account for AMD networking staff use.

Password file format

The standard password file should be implemented on all machines as it comes from the distribution tape, with two modifications:

- 1) An account called localroot should be added with the same UID and GID as root with a locally defined root password. This not only allows emergency access, but another root UID “cron” entry (see Automanagement below) for local cron service.
- 2) The sysdiag login should have a password.
- 3) The root account should not permit logins. (“*” in the password field of /etc/passwd)

2.9. Configuration Control

The Yellow Pages system contains most of the information necessary to run the domain. It does not maintain /etc/printcap, /var/spool/cron/crontabs/*, or associated system administration tasks such as log truncation and sanity checks.

In the spirit of standardization, files should reside on the YP master server and be “distributed” to all the Suns on the network. The tool that provides this update service should be rdist.

The following files should be distributed via rdist from the master YP server:

- 1) /etc/printcap with one or two special versions for any hosts with printers.
- 2) /var/spool/cron/crontabs/root (See Automanagement below)
- 3) /usr/local/adm/{monthly, weekly, daily, hourly, halfhourly} (See Automanagement below)
- 4) /etc/sendmail.cf. Only one version is needed. It redirects any mail with a domain marker other than amd.com to the centralized AMD mail handler for further processing. Long term direction will be toward a nameserver system. The configuration file should properly set the source domain name to amd.com as the default is usually sun.com, arpa, or some other vendor default.

2.10. System Automanagement

Rdist should deliver a default root crontab entry (for the System V cron system in Sun OS 4.0) as well as a set of standard administration scripts that are run monthly, weekly, daily, hourly and halfhourly.

The scripts should function according to the following guidelines:

- 1) `/var/spool/cron/crontabs/root` should be a standard file that calls `/usr/local/adm/{monthly, weekly, daily, hourly, halfhourly}` to perform standardized system management tasks, such as log printing and truncation.
- 2) `/usr/local/adm/{monthly, weekly, daily, hourly, halfhourly}` should contain the generic system administration shell scripts applicable to all hosts.
- 3) Any additional local tasks should be placed in the `/var/spool/cron/crontabs/localroot` crontab file.
- 4) All system log files should be in `/var/log`.
- 5) All printer queuing directories should be subdirectories of `/var/spool/lpd`.

2.11. Root and Remote Access

The root password should be disabled on all systems by replacing the encrypted password with the standard “no password” marker “*”. Any user needing root access should be issued an “su” account and put into the local subdomains `xxx-sysadmin` netgroup. Every quarter “su” privileged accounts should be reviewed, removing any accounts that are no longer needed. If an “su” user leaves the company their account will be disabled immediately.

At no time should a SETUID shell or command interpreter “shortcut” be allowed anywhere in the system allowing root access without a password. If such a file is discovered, disciplinary action should be taken against its true owner.

At no time should a SETUID shell script be allowed. If discovered, it should be disabled and the real owner should be educated in UNIX security practices.

Remote access may be controlled by the `xxx_hosts.equiv` netgroup. The `/etc/hosts.equiv` file should have one line containing the following:

`+@xxx_hosts.equiv`

(Where `xxx` is the subdomain the host belongs to.)

2.12. NFS Access

NFS access must be tightly controlled as “superuser” privileges may be obtained easily by rebooting a client to the UNIX single user mode. The following rules must be enforced by system administrators:

- 1) NFS clients should not be granted “root” write permission, with the exception of exported client root filesystems under Sun OS 4.0. (The client must be able to write to /tmp in its own root filesystem.)
- 2) No filesystem should be exported to all clients by default. Each client should specify all hosts allowed access.
- 3) Read only file systems should be mounted *soft* because soft mounting gives better performance. (*Soft=soft* mode. When an error is detected it is reported to the user and the request fails.) Read only file systems include /usr and /usr/local.
- 4) Read/Write filesystems should be mounted *hard* because writes are critical and must be retried. (*Hard=hard* mode. When an error is detected, the request is repeated until it completes without an error or times out.)

2.13. What Wasn't Discussed.

The subject of distributed system management can fill a book of hints and tips. Topics regarding backups and account management are not discussed, nor are certain site specific security issues regarding dial-in, uucp and outside internet access. These topics are scheduled for discussion at a later date.

3. Conclusion

This document in no way covers all aspects of UNIX system setup and administration, but instead presents a framework upon which to base new system installations. Smaller issues and problems were not dealt with. The presented framework channels smaller problems in the proper direction where solutions appear obvious.

This document reflects a certain philosophy for managing UNIX based workstations. It is a philosophy of commonality for users and system managers. Bringing workstations into conformance with these guidelines is by no means an easy task, but it is important to realize that because these standards exist, the day to day management of UNIX workstations will be practically nonexistent. The time spent on planning and initial setup will be recovered many times over. By writing things down, automating the mundane and moving on to bigger and better tasks you will be rewarded with less downtime, contented users and far less headaches.

Computer Aided Capacity Planning of a Very Large Information Management System

W. Bruce Watson

L-542

Lawrence Livermore National Laboratory
Livermore, CA 94550 (415)422-3756

Project Objectives

In 1986, the Defense Logistics Agency within the Department of Defense embarked upon a system wide modernization program encompassing ~15 sites, which will extend well into the first decade of the 21st century at a total cost ultimately of ~2.5 billion dollars. The TIS program is currently under contract with them to supply a decision support system that will enable managers to manipulate and configure various business functions and data, and then examine the computational consequences of such manipulations. By computational consequences is meant the size, configuration and performance of the hardware components as dictated by these manipulations. The two principal areas of research here concern:

- 1) Characterizing work and its dependence upon hardware in a way that is somehow independent of software design.
 - 2) Configuring hardware and network components based upon the above characterizations in such a way that certain user supplied performance criteria are satisfied.

The objective of this TIS effort is to determine the feasibility of building a model of what basically amounts to a Sears and Roebuck type operation based entirely on relatively abstract characterizations and analyses of business functions and their dependence on generic computational resources.

We see the generalization of this technique to include the sizing and performance evaluation of all types of computing systems as an important secondary goal.

Approach

The model takes as its input, then, a description of the relevant business areas in terms of their functions, existing applications, data bases, and information usages, assiduously avoiding all the details of their current implementation, e.g. specific business practices, or particular application software, hardware and operating systems. This description has the feel and look of an elaborate information flow model; it is purely qualitative.

To quantify the dependence of the various elements of this description on real hardware resources, we first quantify it in terms of what we have termed metaservices. Then, assuming

that the dependence of metaservice upon real hardware is well understood, it is a straight forward matter to complete the transformation of high level business activity into hardware requirements.

We should note at this point that the Defense Logistics Agency itself is responsible for providing the business area descriptions. The Institute for Defense Analyses, also in Washington, D.C., is responsible for quantifying the relationship between the various facets of this description and metaservices. We, in addition to establishing a proof of concept for this modeling effort, are responsible for finding, verifying and validating the numerical relationship between metaservice and real hardware.

We wanted the system (called DOSCAPS) to be modular in design (figure 1), and wherever possible to rely upon off-the-shelf, readily available, vendor supported software and hardware. In the process of sizing, configuring and evaluating the hardware components required by the user's selection and arrangement of business functions and data, the system must cycle through a variety of calculations, e.g., a Mean Value Analysis, a Linear Program, a Knowledge-based evaluation, etc. The DOSCAPS system itself controls the sequence of these calculations which share a common interface implemented as a database management system.

FY88 Progress

The selection of the set of metaservices is of critical importance; they must be complete (i.e., capable of accounting for all of the computational work inherent in each business activity), and orthogonal (i.e., not over accounting any of the work). The first task facing and completed by the three participants in this effort was to identify and describe these metaservices. An example of a metaservice is the standard-query-to-a-database. This metaservice includes all the CPU overhead and disk thrashing required to perform it, but does not include the work required to form or transmit the query, or to transmit the response. These are handled with other metaservices, such as electronic-mail and file-transfer.

The next task completed this year was the design and implementation of a graphical user interface that could demonstrate the proof of concept, as well as be simple enough for MIS professionals to understand and use. A summer student from the New Mexico Institute of Technology has programmed it in C; it is quite impressive. The development environment for this decision support system is a Sun network consisting of a file server and Sun 3/60s relying heavily upon the UNIX OS and Suntools graphics interface. Unify is the database management system.

Mean Value Analysis routines have been added to evaluate the performance of trial configurations.

The proof of concept was successfully demonstrated to and well received by officials and MIS professionals from the Defense Logistics Agency.

FY89 Plans

The big tasks facing us next year will be to:

- 1) Create and embed a knowledge-based system capable of generating trial hardware configurations.

- 2) Compile a set of valid metaservice-to-hardware metrics.
- 3) Refine the user interface.
- 4) Integrate the Unify database into existing DLA data structures.

References

"Quantitative System Performance," Lazowska, Zahorjan, Graham and Sevcik. Prentice Hall, 1984.

System Administration in the Andrew File System

Marybeth Schultz Cyganik

System Manager
Andrew Systems Administration
Carnegie Mellon University

Overview

The Andrew File System (AFS) is a distributed file system running at Carnegie Mellon University. The AFS is location-independent, allowing a user to log into any Andrew workstation on campus and see the same hierarchical file structure. The user does not have to know what file server houses his files, nor does he need to mount any file systems in order to view them.

Most AFS system administration is done with *volumes*. A volume can be thought of as a container, housing a set of related files. For example, each user on the system has their files and directories stored in a single volume. Other groups of related files, such as binaries and program source, are similarly housed in one volume. Although it is not necessary for each volume to contain related files, it aids in administering the system. The volume is the unit for tape backup and restore, replication, quota enforcement, and load balancing on the file servers.

Statistics

The following statistics will give the reader an idea of the size of the Andrew File System at Carnegie Mellon University:

- 7,600 accounts in the password file
- 14 SUN file servers, located in one building, serving the campus community
- 1 SUN file server that serves as control machine, keeping the other file servers in sync
- 25 GBytes of data on disk
- 17,000 volumes
- 400 workstations and 500 PC's attached to the network

The remainder of this paper will focus on the databases and servers which contain information that is vital to the smooth administration of the system.

Current Andrew File System Database Implementation (AFS2)

The System Control Machine (SCM) is vital to the operation of the file servers. The SCM provides file servers with up-to-date copies of binaries and sources, collects the statistics which are monitored by the Operators in order to keep the system running smoothly, and houses the read-write copies of important system files. The SCM is also responsible for migrating read-only copies of these files to the file servers via an update server. In general, the SCM functions to introduce changes into the system, ease system administration, and keep the file servers in sync.

Authentication Database

The authentication database allows users to access their files in a secure fashion. Basically, it is a file containing all of the users' encrypted passwords. All password changes are performed on the read-write copy of the authentication database that resides on the SCM, and is then propagated to the file servers. New users are added to the authentication database during the execution of the *adduser* program. If a user attempts to login to the system, but is not in the authentication database, he may be granted a local login, but he will not be authenticated in the eyes of the AFS.

Protection Database

File protection in the Andrew environment is done differently than in UNIX, giving users more flexibility when sharing files. An *access list* is a list of users and/or groups of users which specifies rights to a directory. These rights include *read*, *write*, *insert*, *delete*, *lookup*, *lock*, and *administration*. In AFS, only directories have access lists. The rights listed in the access list are combined with the Unix "owner" protection bits to control directory permissions. The low-order Unix protection bits are generally ignored.

The file server process uses the protection database (*pro.db*) to interpret access lists on directories. It is usually built by the utilities that add new users to the system, however, it can be generated manually. Every user in the password file and group specified in the groups file must have an entry in the protection database.

Volume Location Database

The Volume Location Database (VLDB) is a file that describes every on-line volume in the system. The VLDB provides addressing information to the workstation, and associates read-only replicas with their respective read-write volume. The VLDB lives on the control machine, and is updated by polling each of the file servers for a list of the volumes it houses. Each of these lists is copied over the network to the control machine, where a master list is then built. Finally, the control machine then propagates a copy of the new master list back to each file server. All updates to the VLDB are done on the control machine. A rebuild of the VLDB can be forced manually, however it is typically run out of *cron* once each hour.

Limitations of AFS2 Implementation

Due to the nature and size of the Andrew File System, the System Operators often have to know too much to get routine tasks done. System administration and operation tools are difficult to use, and sometimes do not exist. As a result, system performance and reliability occasionally suffer. In addition, the key databases (authentication, protection, and VLDB) do not see changes

immediately. It can take up to several hours for any given change to propagate to all database copies throughout the system. Depending on the nature of a given problem, this time lag can often have serious consequences.

Implementation of Future Andrew File System Databases (AFS3)

The upcoming AFS implementation will improve reliability and performance, and will also provide the tools and mechanisms to make the system more manageable. It is built upon the *ubik* replicated database package, which will greatly ease administration. Updates to important system files will be reflected immediately to all their replicas, thus eliminating the need to wait for things to propagate throughout the system.

The Nanny Process

The Nanny process will effectively replace the role of the SCM, and distribute its work amongst the individual file servers. Nanny's responsibilities include ensuring that the appropriate processes are running on each file server (restarting them if necessary), ensuring consistent and up-to-date binaries and sources, providing central and local logging of statistics and error information, and maintaining a database describing processes local to each file server.

Authentication Server

The authentication server is compatible with the Kerberos system designed at MIT's Project Athena. This server will be responsible for verifying a user's identity and generating tokens used for this purpose. Unlike the AFS2 authentication database, updates will propagate immediately. Thus, there will not be a waiting period when new users are added to the system or when passwords are changed.

Protection Server

The protection server will determine if a certain user or group of users have rights to the specified directories. The protection database *pro.db* will be updated immediately without waiting for new generations to be built.

Volume Location Server

In addition to maintaining the Volume Location Database (VLDB), the Volume Location Server will provide the mechanisms necessary to administer the system volumes. Volume creation, deletion, replication, and relocation operations will be visible immediately throughout the system.

Further Information

Although AFS3 is still under construction, further information will be provided by sending electronic mail to info-afs@andrew.cmu.edu.

Service Management at Project Athena

Daniel E. Geer, Jr.

Project Athena

Massachusetts Institute of Technology

Cambridge, Massachusetts 02139

geer@ATHENA.MIT.EDU

1. The Problem

Only a network services model of computation permits the unlimited upward scalability of the workstation computing environment. Yet, as the network services environment grows, both in numbers of nodes and in breadth of the service offering, the complexity of the service to server mapping grows, too. At each order of magnitude of network extent, new demands on centralized, authoritative, reliable service management arise. Fiscal restraint as well as physical limitations indicate that the solution must grow in costs at rates far less than linear with the number of workstations.

2. The Solution

The purpose of the Athena Service Management System (SMS) is to provide a single point of contact for authoritative information about resources and services in a distributed computing environment. The requirements for the initial Athena SMS include:

- Management of 15,000 accounts, including individual users, course, and project accounts, and special accounts used by system services.
- Management of 1,000 workstations, timesharing machines, and network servers, including specification of default resource assignments.
- Allocation of controlled network services, such as creating and setting quotas for new users' home directories on network fileservers, consistent with load-balancing constraints.
- Maintenance of other control information, including user groups, mailing lists, access control lists, etc.
- Maintenance of resource directories, such as the location of printers, specialized file systems (including privately supported file systems), and other network services.

SMS is a centralized data administrator providing network-based update and maintenance of system servers. This must be accomplished with the utmost robustness.

- Conceptually, SMS provides mechanisms for managing servers and resources. This aspect comprises the fundamental design of SMS.
- Economically, SMS provides a replacement for labor-intensive hand-management of server configuration files.
- Technically, SMS consists of a database, a server and its protocol interface, a data distribution manager, and tools for accessing and modifying SMS data.

SMS provides coherent data access and data update. Access to data is provided through a standard application interface. Programs designed to reconfigure network servers, edit mailing lists, manage group membership, etc., all use this application interface. Applications used as administrative tools are invoked by users; applications that update servers are (automatically) invoked at regular intervals. Management reports and operational feedback are also provided.

Concurrent Access Licensing and NLS

David Ortmeyer

Section Manager
Domain Distributed Systems
Apollo Computer
330 Billerica Road
Chelmsford, MA 01824

A key challenge facing system administrators of large heterogeneous distributed networks is the provision of efficient, transparent user access to the system's software resources. Users need to be able to access these resources from any point in the network transparently. Equitable and efficient access presupposes that these resources can be placed where they are needed, when they are needed, and that they are neither under- nor over-utilized.

Licensing arrangements for this software should be sufficiently flexible and cost-effective to support the system administrator's endeavors. License enforcement schemes which tie software to particular nodes in the network (i.e. nodelocking), interfere with equitable and efficient distribution of software resources and the provision of easy access to those resources. Licensing arrangements which require companies to pay far more for software than usage patterns dictate in order to realize the benefits of flexible placement (i.e. site licensing) are unnecessarily punitive.

Concurrent access licensing offers an alternative model to nodelocking and site licensing. The concurrent access model, by regulating the number of users that can simultaneously access an application, provides for maximum flexibility and a close mapping of software licensing charges to actual usage patterns. Software which is concurrently licensed may run on any system networked to a license server responsible for managing licenses for the application. When an application is invoked, it seeks out a license server which manages its licenses, requests one and (if granted) runs. If a license is not available, the application may place the user on a queue managed by the license server.

Use of a concurrently licensed application depends neither on the location of the application nor the location of the user. In other words, the system administrator can distribute applications and licenses for those applications in a relatively unconstrained manner. Moreover, concurrent licensing ties the licensing fee to the concurrent access rate, a far better mapping to actual network-wide use than either site or nodelocked licensing schemes.

Apollo's Network Licensing System is an example of a portable distributed software licensing management system which supports the concurrent access licensing model. It can accommodate a wide range of software applications by offering application vendors a compact but flexible library of licensing calls that can be used to implement a large variety of run-time policies. The system also offers system administrators at end-user sites a range of powerful administration tools which

manage licenses, control user access to applications, and monitor and report on software usage patterns (historically and in real-time).

The Network Licensing System is built on Apollo's Network Computing System (NCS) and is designed to scale easily and interoperate in heterogeneous networks of Apollo, Sun/3 and Vax(VMS) platforms. Future targets include IBM(MSDOS), IBM(OS/2), and Vax(Ultrix). It can be ported to any communications substrate offering a datagram facility.

The Network Licensing System consists of four architectural components, combined for sale into two packages, one for the Application Vendor (LSLOCK) and the other for the System Administrator (NLS). The LSLock package (so named because it offers a means of "locking" the application from unauthorized use) consists of:

1. An application library of licensing calls used to instrument the application to be licensed. The application binds with this library. The library has the responsibility for locating the server and making all of the actual RPC calls to the server.
2. A license creation tool which generates an encrypted string representing a "license record". This license record is transmitted to the System Administrator at the end-user site and contains information on the start and expiration dates of the licenses, the number of licenses and a special field containing any license-specific information that the application needs to consider at invocation time to determine how to run under the license. To prevent illegal duplication of the license record at the end-user site, the record is tied to a particular license server running on a particular platform.

The NLS (Network License Server) package consists of:

1. The license server which manages the license database. The server listens on a DDS or IP socket and communicates with its clients, the applications and the administration tools via remote procedure calls. The underlying RPC mechanism is provided by NCS. The server supports two interfaces, the application licensing call interface mentioned above, and a set of operations for the administration tools. Each operation records information in a log file that can be examined by System Administrators to determine usage patterns and error conditions.
2. A set of administration tools which access the license database (insert, delete), show licensing status and create special reports.

A Subscription-Oriented Software Package Update Distribution System (SPUDS)

Ola Ladipo

AT&T Bell Laboratories
2000 N. Naperville Road,
Naperville, Illinois 60566-7033
(312)979-4705

ihlpa!ola@att.att.com
att!ihlpa!ola

ABSTRACT

This paper describes a subscription-oriented Software Package Update Distribution System (SPUDS). The system provides secure file distribution over existing networks, using a client-server view for the package software. The server systems hold official versions of selected packages and act as software distributors, while the client systems (subscribers) are software recipients. No particular networking medium is assumed; however, the network must provide file transfer and remote execution.

This system is beneficial to any UNIX® system administrator that distributes software over any network which provides file transfer and remote execution.

1. Site Characteristics

SPUDS was designed to meet the needs of AT&T Bell Laboratories Computation Centers concerned with software distribution across multiple systems. The computation centers are located in Colorado, Illinois, New Jersey, and Ohio. The machines that the centers use range from workstations to large mainframes interconnected by various network media. At the Illinois location alone, two to five terabytes of data are moved over these networks each month.

2. Problems

Listed below are a few of the problems that SPUDS was designed to solve:

- Managing software on a cluster of heterogeneous machine types over batch networks is often a repetitive and time consuming job. Updates sometimes get lost and unwanted updates (either from a friendly administrator or intruders) are sometimes received and installed without the administrator's knowledge.
- At AT&T Bell Laboratories Computation Centers, many updates are generated frequently. To manage the installation of these updates, people developed their own independent

procedures. This has resulted in many different procedures being used to solve the same problem. This leads to duplication of effort and moving from center to center requires learning new procedures.

- When errors occur during the installation of updates, systems are sometimes left in an inconsistent state owing to lack of back-out procedures.

software distribution mechanism that allows individual selection of installed software.

3. Solution

This section is divided into two parts. The first part gives a general description of SPUDS, and the second part describes the important SPUDS features.

3.1. Description

SPUDS was designed as a subscription-oriented software package distribution system. In this scheme, the logical unit of distribution is a **package**. A package is a collection of files and file attributes. Basically, three major players are involved in this scheme. The **package provider** controls the contents of a package and is usually the author or the maintainer of the package. **Subscriber machines** receive or subscribe to software packages via the SPUDS system. The **package server**, one or more machines, keeps the official versions of selected packages. A package server can also be a subscriber machine. Each subscriber machine receives updates from the package server. When a package provider updates a package on the server machine, the server advertises the update package to all the subscribers. Advertising uses a three-way communication method. When a package server receives a new update package, it sends a message to all subscribers. On receiving the message, each subscriber machine uses the message and the information in these databases to determine if the package update should be requested:

- **package catalog:** This catalog contains information about requestable and installed packages. The package catalog always reflects the local or subscriber's view of all available packages. The package server catalog contains the most complete view of all available packages.
- **package list:** For each package already installed on the machine, there is a corresponding package list database. The package list database contains information about each file in a package.

The message and the database information allow the subscribers to determine if a software is out-of-date, up-to-date, or not needed. Only the subscriber machines that want the update send a request back to the package server; the rest discard the message. On receiving the request, the package server sends the update package to the requested subscriber.

3.2. What SPUDS Provides

SPUDS provides these features:

Flexible distribution - The flexible distribution allows subscribers to decide which updates to accept. The package server is not allowed to "force-feed" update to the

subscribers; the server role is to advertise the updates and let the subscribers decide when to request the updates. This flexibility also allows for excellent software synchronization. A subscriber can send an audit request to the package server for a consistency check against the server's official base. If an inconsistency is detected, the subscriber is informed and a request for the official update is generated.

Robustness - The subscriber can automatically request an update without administrator intervention. The administrator has an option of being notified at every update. Because updates are sent only to subscribers that request them, subscribers do not receive duplicate updates and data storage resources are optimized.

Security - In any software distribution and installation package, security is important, and SPUDS is no exception. Each package on the server machine has a **password** known only to the current provider of the package. It permits the provider to update the package on the package server machine. Each machine also has a **keyfile** database. Each keyfile contains a key that the server and the subscriber use to identify one another. All information sent between the package server and the subscribers contains checksums that are used to detect package tampering. Critical data can be encrypted and/or compressed. System administrators are informed of any attempts to update software not previously requested and they are able to restrict software requests and/or installation.

Distributed administration - SPUDS reduces the need for administrator intervention. It maximizes local administration control by letting the local administrators determine which software gets installed and who can access the software.

Ease of administration - All updates are install in an atomic fashion; if an error occurs during installation, the update is backed out. Distribution and installation are separable processes allowing system administrators to spool an update for review before installing. Administrators can automatically request updates and a lost update is not that serious. If a machine misses an update, it can automatically request a replacement. The server system supports arbitrary numbers of updates and target processor types. A cluster of heterogeneous machine types can be updated from one server. Administrators can audit currently installed updates for consistency against the server's official base.

4. Acknowledgements

Alan Hastings and Andy Schnable, both of AT&T Bell Laboratories are co-authors of the requirements document and were involved in reviewing the design document for *SPUDS*. Alan also built a prototype that was used in a controlled environment. Alan, Andy, and Dorothy Ann Lash (AT&T Bell Laboratories) reviewed this document.

System Administration and Maintenance of Fully Configured Workstations

Robert E. Van Cleef

Workstation Subsystem Manager for Operations
General Electric Corporation*
NASA Ames Research Center
Moffett Field, CA 94035

ABSTRACT

NASA's Numerical Aerodynamic Simulation project (NAS) supports over eighty UNIX† workstations, the majority of which are high end graphics workstations used by scientific researchers. This paper discusses the administration tools and procedures used to maintain the system files on a large number of fully configured workstations.

Overview

NASA's Numerical Aerodynamic Simulation Program (NAS), at the Ames Research Center in Moffett Field, California, has three main objectives:

- Provide a national computation capability available to NASA, DoD, industry, and other government agencies and universities, as a necessary element in ensuring continuing leadership in Computational Fluid Dynamics (CFD) and related disciplines.
- Act as a pathfinder in advanced, large-scale computer system capability through systematic incorporation of state-of-the-art improvements in computer hardware and software technologies.
- Provide a strong research tool for the NASA Office of Aeronautics and Space Technology.

An important aspect of the work at NAS is the use of Graphics Workstations as a tool for Computational Fluid Dynamics research. This has led to the installation of over forty Silicon Graphics Iris workstations for use by local researchers and programmers.

This paper will present an overview of support policies, configuration standards and some of the tools that are used by the workstation support team. Although this paper primarily addresses workstation issues, many of the concepts and policies mentioned in this paper are also used in the administration and maintenance of the larger NAS systems (see Table 1).

*This work was supported by contract NAS 2-11316 of the National Aeronautics and Space Administration to General Electric Corporation.

† UNIX is a Trademark of Bell Laboratories.

Table 1: Current NAS computer systems

Quantity	System Type	Version of Unix
2	Cray 2	UNICOS 4.0
1	Cray YMP	UNICOS 4.0
1	Amdahl 5880	UTS 1.2.2
4	DEC VAX 11/780	BSD 4.3
8	Silicon Graphics IRIS 3030	SGI GL2-W3.5
9	Silicon Graphics IRIS 3130	SGI GL2-W3.5
25	Silicon Graphics IRIS 2500T	SGI GL2-W3.5
15	Silicon Graphics IRIS 4D/60	SGI 3.2
11	Sun 3/50	SunOS 3.5
7	Sun 3/260	SunOS 3.5
6	Sun 3/280	SunOS 3.5

Levels of Support

Each workstation is given a certain level of support after consulting with the owner/users of the system. The level of support is directly related to the use of the system and level of operating system control the user desires. It is understood that the owner/user of a system may reject any level of restriction -- such as the restriction on the distribution of *superuser* passwords, but in doing so, they also reject the corresponding level of support.

There are four classifications of systems; fully supported (FS), modified for application development (MAD), systems application development (SAD), hardware only supported, and engineering development (HOSED). The differences in the levels of support are in what services the workstation support team will NOT supply, and what flexibility the users are allowed in modifying the system files. A listing of what types and numbers of systems are in each category, Table 2, shows that the majority of the users have elected to trade flexibility for reliability and support.

Table 2: Workstations in each support category

Type of System	FS	MAD	SAD	HOSED
SGI IRIS 2500T	17	4	2	2
SGI IRIS 3030/3130	13	4		
SGI IRIS 4D	4	10	1	
SUN 3	20		1	3

Configuration of NAS Workstations

This section will explain the standards used to configure the NAS workstations, including: standardization of file system structure, file backup and recovery, user's home directories, and standardization of disk partition sizes and the use of symbolic links to maintain identical system images across all of the workstations in one class.

Workstation Support Tools

One of the most time consuming aspects of system administration is the maintenance of the file system. Besides the traditional running of *fsck* on reboot, the system administrator has to insure that unneeded files are deleted, log files are kept small, and that the system files remain intact, with the correct permissions and ownership. This section discusses some of the tools that we use to simplify these tasks.

There are tools that we use to automate and simplify the file system maintenance and cleanup, including: the *skulker* -- nightly file deletion, *Archive.logs* -- nightly aging of log files, *rdist* -- clone file systems across the network, *trsh* -- run *csh* on multiple systems simultaneously, *binaudit* -- track changes in the system files, *perms* -- maintain ownership and permissions of system files,

There are tools that simplify the routine system administration tasks, including: *addacct* -- add and delete accounts, *time.sh* -- check and update system time from a remote system, *lsu* -- enhanced versions of '*su*', *serverd* -- switch file servers if one goes down, *userval* -- track and validate user entries in password files, and *netmotd* -- maintain and update the *motd* files.

There are tools that we use to allow a non-privileged user to do privileged functions, eliminating the need for intervention by a system administrator, including: *usrboot* -- reboot the system correctly, *nfsmount* -- use NFS to mount/unmount user files on other workstations, *sob* -- simple operator backup utility, and *gethome* -- find the '*~*'ome directory of any user.

Acknowledgements

I would like to acknowledge the support of the NAS operations and development workstation teams, and especially Michele Crabb and Matt Bishop.

Capacity Testing a HYPERchannel-Based Local Area Network

W. Bruce Watson

L-542

Lawrence Livermore National Laboratory
Livermore, CA 94550 (415)422-3756

ABSTRACT

Given the amount of resources they have invested in their local area network (LAN), network managers at Lawrence Livermore National Laboratory have an abiding if not critical interest in determining its remaining life span. Using a hardware monitor, researchers at LLNL have tested the capacity of their mature (over 10 years old), heavily loaded (5 Crays, 3 Ethernet gateways, 13 terminal concentrators, 5000 terminal, 40 node) two-trunk, HYPERchannel based, high data rate LAN. Results indicate that the current load could be intensified by 40% before the network reaches saturation. Further, as the load approaches saturation, overly persistent, low-level protocols begin thrashing, effectively destabilizing the network and destroying much of its potential capacity. Finally, it is clear what steps need to be taken to extend its life span and improve its performance under heavy loads.

Introduction

System designers at the Lawrence Livermore National Laboratory (LLNL) are currently deep in the throes of designing, implementing and debugging a new, distributed, client/server, message passing operating system, NLTSS. The underlying communications software and hardware comprise one of the structures that play a critical role in overall system performance. To aid them in achieving optimum network performance, systems designers at LLNL have acquired a hardware monitor for their Network Systems Corporation (NSC), HYPERchannel-based, high data rate local area network (LAN). This monitor, the HYPERtools Inc. Enhanced HYPERchannel Monitor Device (EHMD), copies the headers of all frames traversing every HYPERchannel trunk into internal buffers. Simultaneously, the EHMD's higher level 80386-based processor analyzes the data in these buffers to reveal the network's performance, e.g. message size, delay, and interarrival time distributions, source/destination path activity, utilization, throughput, etc. The HYPERchannel is NSC's 50mbps, baseband LAN product utilizing CSMA with collision avoidance and capable of interconnecting a plethora of diverse computers and peripherals.

In 1984, I concluded a study sponsored and funded by the National Bureau of Standards to characterize the traffic in LLNL's HYPERchannel-based network using these same and other metrics; an early prototype of the EHMD proved indispensable..

It was the data in this report that led ultimately to the current study. In comparing the current

intensity of network activity to that reported in the 1984 study, I observed that a dramatic increase had taken place. In August, 1987, I observed peak afternoon traffic as high as ~350 messages/second, whereas in 1984, the overall afternoon network traffic had been ~150 messages/second. During this same interval, the network had swelled to 40 nodes (adapters) from 25 nodes. I began to wonder how much network capacity remained, and how best to use the EHMD to measure it. Given that we rely heavily upon this network, and have invested considerable money and effort in it, we intend to prolong its useful lifespan and to extract every nickle's worth of value and every line of coding's worth of sweat from it.

Capacity Testing

Consider some arbitrary and small moment of network time containing some network traffic (characterized by its message sizes and interarrival times, and their source/destination pairings). Suppose we had at our disposal the means to deny, in varying amounts, some fraction of that moment to the traffic normally present in that moment. We could then gradually increase the fraction of network time denied to normal traffic to the point where not enough time remained to accommodate that traffic. Suppose that by such a process we discover that the network collapses if we deny more than 40% of the available time. We could conclude that the current load consumes no less than 60% of the current capacity.

One way of denying time to the real load is to consume it with artificial load, but the artificial load whatever it is and however it is generated must not perturb the real load in any way whatsoever. It can not alter receiver/sender availability, or the true load's interarrival rates or message lengths. Note that for the purpose of capacity testing, we do not need to know the true load's characteristics; we need only to force it to fit in less available time.

For the purposes of our capacity test, we relied on two Amdahl 5840s which recently had been attached to the network but which could not be in service until their network interfaces had been debugged, i.e., no other node in the network knew of their presence. The capacity test was conducted during the two weeks between the successful debugging of their network drivers and their full integration into the network. The capacity test proceeded as follows: each afternoon from 1pm to 5pm, the network was subjected to a known load test load. That is, one Amdahl would transmit messages of known length, mode and intensity over the network to its counterpart. We used the EHMD to verify that the load was in fact what we wanted it to be. Once we had a means of generating a known load, next we needed some means of measuring the network's response to that load. We decided upon the following metrics:

- 1) Idle time which in essence is the complement of the utilization.
- 2) Transit delay which is more or less proportional to the transmission delay.
- 3) Dither which is a kind of protocol thrashing peculiar to HYPERchannel-based networks.

The complete explanation of these metrics is sufficiently complex to warrant a fuller treatment which is to be found in the complete report which I am sending via US Mail.

Results

It appears that the current characteristics of network load at LLNL are such that:

- 1) Our two trunk HYPERchannel can never be less than ~20% idle (80% utilized) and still be productive.
- 2) During peak times, we currently operate at ~57% of our network's capacity with an average transit delay of ~4 milliseconds.
- 3) Our second trunk is critical to successful network operation during peak loads, and that it no longer provides us with the redundancy we require.

We must distinguish between "percent of capacity" and "utilization." As long as there is no significant change in its other characteristics, the applied load can be increased 75% before the network will saturate ($.57 \times 1.75 = 1.00$). At this point the applied load will consume ~100% of available network capacity, and the network will be 80% utilized.

Our network can function under a 43% test load albeit not without some difficulty, i.e., average transit delays are ~3 times normal. The implication is that as our real load increases, the transit delays will increase up to a maximum average value of ~10 milliseconds. Our results indicate that as the test load is increased beyond 43% and approaches the 49% mark, the network will collapse abruptly because of dither. And indeed, when subjected to the 49% testload, the network collapsed so quickly due to dither we were unable to collect any idle time or transit delay data.

Conclusions

The HYPERchannel-based LAN which constitutes the communications backbone for the Livermore Computer Center at LLNL currently operates at ~60% of its capacity during periods of peak load (e.g. Friday afternoons). Experiments predict that as the network load approaches its maximum limit, its interaction with overly persistent protocols at the trunk, link and possibly higher layers will cause the network to collapse abruptly. The maximum sustainable load is not easily characterized as some fraction of the HYPERchannel's raw bandwidth or maximum message rate.

USENIX Association Services and Benefits

The USENIX Association is a not-for-profit organization of individuals and institutions with an interest in UNIX and UNIX-like systems and the C programming language. It is dedicated to fostering the development and communication of research and technological information and ideas pertaining to UNIX and UNIX-related systems. The Association sponsors workshops and semiannual technical meetings, produces and distributes a bimonthly newsletter, *;login:;*; publishes a quarterly technical journal, *Computing Systems*; and serves as coordinator of a software exchange via its "Software Distribution Tapes."

The Association was formed in 1975 and incorporated in 1980 to meet the needs of UNIX users and system maintainers who met periodically to discuss problems and exchange ideas concerning UNIX. It is governed by a Board of Directors elected biennially.

The USENIX Association offers several services to its members:

- Mailing of the newsletter *;login:;*;
- Mailing of the technical journal *Computing Systems*;
- Offering of various UNIX publications and technical information for purchase;
- Presentation of technical meetings twice a year and single-topic workshops periodically;
- A discount on the meeting registration fee;
- The right to order 4.3BSD UNIX Manuals;
- The right to vote on matters affecting the Association, its bylaws, and in the election of its directors and officers.

STUDENT MEMBERSHIP \$15

Open to any full-time student at an accredited educational institution. A copy of your student I.D. card must be provided.

INDIVIDUAL MEMBERSHIP \$40

Open to any individual or institution. Individual Members may vote; however, they do not automatically receive the Distribution Tapes or other services requiring UNIX license verification.

USENIX Association Membership Application

Membership is by Calendar Year

Please type or print

[] New [] Renewal

Name: _____

Address: _____

Phone: _____

uucp network address: *uunet!* _____

Individual, Corporate, and Supporting categories are all open to either institutions or individuals.
Membership fees are:

[] \$ 40 Individual

[] \$ 15 Student (full-time)
With copy of student I.D. card

[] \$ 275 Corporate

[] \$125 Educational Institution

[] \$1000 Supporting

[] Check enclosed: \$ _____

Payments must be in US dollars payable on a US bank

[] Purchase order enclosed; invoice required

[] Check if you do NOT want your name and address made available to other members.

[] Check if you do NOT want your name and address made available for commercial mailings.

Please complete and return this form with
your purchase order or payment to:

USENIX Association
P. O. Box 2299
Berkeley, CA 94710

For Office Use

Inst:

Mem#: Check #:

Lic: Rf:

Date: Db:

USENIX Conference & Workshop Proceedings Order Form

Copies of the Proceedings of the USENIX-sponsored conferences and workshops listed below are available from the Association office.

Ship to:

Mail one copy of this order form with your check to:

USENIX Association
P.O. Box 2299
Berkeley, CA 94710

(415) 528-8649

How Many	Meeting	Date	Price Each	TOTAL	Overseas Postage Each	TOTAL
____	Large Installation Systems Admin. Workshop	Nov. '88	\$ 8	\$ _____	\$ 7	\$ _____
____	C++ Conference	Oct. '88	20	\$ _____	15	\$ _____
____	UNIX and Supercomputers Workshop	Sep. '88	20	\$ _____	15	\$ _____
____	UNIX Security Workshop	Aug. '88	5	\$ _____	7	\$ _____
____	San Francisco Conference	Jun. '88	20	\$ _____	25	\$ _____
____	C++ Workshop	Nov. '87	20	\$ _____	25	\$ _____
____	Graphics Workshop IV	Oct. '87	10	\$ _____	15	\$ _____
____	Washington DC Conference	Jan. '87	10	\$ _____	25	\$ _____
____	Graphics Workshop III	Dec. '86	10	\$ _____	15	\$ _____

Total cost of copies \$ _____
7% Sales tax (Calif. residents only) \$ _____
Total cost of overseas postage \$ _____
Amount enclosed \$ _____

Shipping Information

- Fourth class mailing to the United States and Canada is included in the price listed. First class and UPS shipping are available; contact the office for more information.
- Overseas mailing charges are per copy. Delivery by air printed matter takes 10-14 days.

Payment Information

- Make checks payable to "USENIX Association" in US dollars payable at a US bank.
- Payment MUST be enclosed with this order form.

USENIX Association

P.O. Box 2299

Berkeley, CA 94710